FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Manipulador robótico para poda automática (Projeto RoMoVi)

José Nuno Gomes de Brito

Relatório realizado no âmbito da unidade curricular de Preparação da Dissertação do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Pedro Luís Cerqueira Gomes da Costa (FEUP)

Co-orientador/Supervisor: Filipe Baptista Neves dos Santos (INESC TEC)

Co-orientador: José Luís Sousa de Magalhães Lima (IPB)

27 de Julho de 2018

Resumo

A comercialização do vinho do Porto é uma das atividades mais importantes e lucrativas do país. As vinhas que se desenvolvem pelas encostas do Douro requerem mão-de-obra especializada, muitas vezes escassa. Uma das atividades vitivinícolas mais importantes é a poda que, quando criteriosamente realizada, maximiza simultaneamente a colheita e a qualidade da matéria-prima, a uva.

A poda, que requer profissionais qualificados e experientes, poderá a prazo ser realizada por sistemas robotizados inteligentes e autónomos. Um veículo autónomo, designado de AGROB V16, dotado de visão e inteligência artificial, será capaz de se deslocar pela vinha e identificar e cortar as vides. Tal veículo é dotado com um braço robótico em cuja extremidade se situa a ferramenta de corte, vulgarmente designada por tesoura de poda.

Um sistema com esta complexidade pode ser subdividido em dois subsistemas principais, sendo o primeiro associado à perceção da vinha e identificação das vides a podar, e o segundo relacionado com o planeamento de trajetórias a realizar pelo braço robótico, com o intuito de efetuar o corte das vides.

O foco desta dissertação incide exclusivamente no planeamento de trajetórias do braço robótico, sendo conhecidos os pontos de corte, previamente fornecidos pelo subsistema de perceção e identificação das vides a podar.

Esta dissertação apresenta uma solução para a problemática do planeamento de trajetórias orientado para a poda. A solução obtida assenta no teste de 23 diferentes algoritmos de planeamento de trajetórias e apresenta a melhor combinação de algoritmos e respetivas configurações, que permitam a realização dos movimentos exigidos, tendo em conta dois critérios de análise: (1) a minimização do tempo para encontrar uma trajetória e (2) a maximização da taxa de sucesso no seu planeamento de trajetórias que não colidam com a vinha, ou seja, a sua assertividade.

Para efetuar os testes dos algoritmos de planeamento, foi montado um ambiente de simulação contendo o modelo da vinha e a plataforma que irá operar no ambiente real, o AGROB V16. Na extremidade do braço robótico foi montada uma tesoura de poda elétrica, que funciona como o seu *end effector*.

Foram realizados cinco testes com um modelo de vinha dito simples e um teste com um modelo de vinha complexo. Os testes foram permitindo a seleção dos algoritmos com melhores resultados em termos temporais e de assertividade.

Após a realização de todos os testes e avaliação dos respetivos resultados, é possível concluir que a combinação dos algoritmos *LazyPRMstar* (para o planeamento desde a posição base do manipulador até à posição de corte) e *BiEST* (para o planeamento desde o ponto de corte até à posição base do manipulador), com o valor de 1cm no parâmetro de planeamento *longest_valid_segment_fraction*, apresenta os melhores resultados em termos de tempo e de assertividade, para aplicações de vinha. Para esta combinação e configuração específica, é de esperar um tempo médio total de 6,456 segundos para o planeamento da trajetória, garantindo com 90% de certeza que as trajetórias calculadas não colidem com a vinha.

Testes em ambiente real não foram realizados devido à indisponibilidade da tesoura de poda no laboratório. Ainda assim, foi desenvolvido o código que permite o planeamento de trajetórias no braço robótico da plataforma real.

Abstract

The commercialization of Port wine is one of the most important and profitable activities of the country. The vineyards that grow along the slopes of the Douro river require specialized labor, often scarce. One of the most important wine activities is the pruning of grapevines that, when carefully carried out, simultaneously maximizes the harvest and the quality of the raw material, the grape.

Pruning, which requires qualified and experienced professional, can eventually be performed by intelligent and autonomous robotic systems. A stand-alone vehicle, called AGROB V16, equipped with vision and artificial intelligence, will be able to travel through the vineyard and identify and cut the vines. Such vehicle is provided with a robotic arm at the end of which is the cutting tool, commonly known as pruning shears.

A system with this complexity can be subdivided into two main subsystems, the first one associated with the perception of the vineyard and identification of the vines to be pruned, and the second related to the planning of trajectories to be performed by the robotic arm, with the intention of making the cut of the vines.

The main focus of the dissertation in exclusively the trajectory planning of the robotic arm, with the cutt-off points being provided by the subsystem of perception and identification of the vines to be pruned.

This dissertation presents a solution to the problem of trajectory planning oriented for pruning. The solution obtained is based on the test of 23 trajectory planning algorithms and presents the best combination of algorithms and respective configurations, allowing the required movements to be performed taking into account two criteria of analysis: (1) the minimization of the time to taken to find a trajectory and (2) the maximization of the success rate of planned trajectories that do not collide with the vine, that is, its assertiveness.

In order to carry out the tests of the planning algorithms, a simulation environment was built, containing the model of the vineyard and the platform that will operate in the real environment, the AGROB V16. At the end of the robotic arm was mounted an electric pruning shears, which functions as its end effector.

Five trials were carried out with a simple vine model and a test with a complex one. The tests allowed the selection of algorithms with better results in terms of time and assertiveness.

After performing all the tests and evaluating their results, it is possible to conclude that the combination of the algorithms LazyPRMstar (for planning from the manipulator base position to the cut-off position) and BiEST (for the planning from the cut-off point to the base position of the manipulator), with a value of 1cm in the planning parameter longest_valid_segment_fraction , presents the best results in terms of both time and assertiveness for vineyards applications. For this combination and configuration, a total average time of 6,456 seconds is expected for the trajectory planning of both trajectories, guaranteeing with 90% certainty that the calculated trajectories do not collide with the vine.

Test in real environment were not performed due to the unavailability of the pruning shears in the laboratory. Even so, the code that allows the planning of trajectories in the robotic arm of the real platform was developed.

Agradecimentos

Em primeiro lugar quero agradecer à minha família, às quatro fantásticas pessoas que vivem comigo, por todas as oportunidades e apoio que me deram ao longo não só do meu percurso académico mas também de toda a minha vida. São o meu maior orgulho e espero um dia poder retribuir tudo aquilo que fizeram e fazem por mim.

Agradeço ao meu avô Joaquim de Brito pelos conhecimentos que me transmitiu acerca das vinhas e que despertaram o meu interesse nesta área.

Quero também agradecer ao resto da minha família por todo o carinho que sempre me deram e por me terem permitido crescer no ambiente fantástico que sempre tivemos, sou certamente uma melhor pessoa hoje devido aos valores que me foram passados ao longo de todos estes anos.

Um obrigado a todos os amigos que fiz na faculdade e que de uma forma ou de outra tornaram estes 5 anos mais proveitosos. Obrigado pelas noites de trabalho, por puxarmos uns pelos outros e por termos crescido e aprendido juntos. Um agradecimento especial ao meu amigo Vasco Agante pela pessoa que é e pelos momentos partilhados nesta faculdade, no Erasmus e dentro e fora do campo.

Agradeço também à minha namorada Catarina Lacerda por todas as palavras de apoio, por não me ter faltado nenhuma vez e por ser a pessoa espetacular que é. Ainda que não tenha acompanhado todo o meu percurso académico, sempre me fez sentir como se estivesse aqui desde o meu primeiro dia nesta faculdade.

Deixo também uma palavra de agradecimento aos meus três orientadores, os professores Pedro Costa, Filipe Neves dos Santos e José Lima, pela oportunidade que me deram de participar num projeto inovador e pelo constante apoio prestado.

Por último, quero agradecer aos meus colegas de laboratório pelo bom ambiente que sempre proporcionaram. Destaco o Luís Santos e o Samuel por toda ajuda prestada e pela sua incansável paciência e ainda o Ricardo Reis, o Duarte Rodrigues e o Filipe Azevedo pelo seu companheirismo.

José Brito

"Automation is not our enemy... Automation can be the ally of our prosperity."

Lyndon B. Johnson

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

FEUP	Faculdade de Engenharia da Universidade do Porto
IPB	Instituto Politécnico de Bragança
INESC TEC	Instituto de Engenharia de Sistemas e Computadores - Tecnologia e Ciência
AGV	*Automated Guided Vehicle*
ROS	*Robot Operating System*
RoMoVi	Robô Modular e cooperativo para Vinhas de encosta
SCARA	*Selective Compliance Assembly Robot Arm*
DOF	*Degrees of freedom*
OMPL	*Open Motion Planning Library*
PRM	*Probabilistic RoadMaps*
LazyPRM	*Lazy Probabilistic RoadMaps*
*PRM**	*Probabilistic RoadMaps star*
*LazyPRM**	*Lazy Probabilistic RoadMaps star*
SPARS	*SPArse Roadmap Spanner algorithm*
SPARStwo	*SPArse Roadmap Spanner algorithm two*
RRT	*Rapidly-exploring Random Trees*
RRTConnect	*Rapidly-exploring Random Trees Connect*
*RRT**	*Rapidly-exploring Random Trees star*
LBTRRT	*Lower Bound Tree Rapidly-exploring Random Tree*
TRRT	*Transition-based Rapidly-exploring Random Tree*
BiTRRT	*Bi-directional Transition-based Rapidly-exploring Random Tree*
EST	*Expansive Space Trees*
BiEST	*Bi-directional Expansive Space Trees*
ProjEST	*Projection Expansive Space Trees*
SBL	*Single-query Bi-directional Lazy collision checking planner*
KPIECE	*Kinematic Planning by Interior-Exterior Cell Exploration*
BKPIECE	*Bi-directional Kinodynamic motion Planning Interior-Exterior Cell Exploration*
LBKPIECE	*Lazy Bi-directional Kinodynamic motion Planning Interior-Exterior Cell Exploration*
STRIDE	*Search Tree with Resolution Independent Density Estimation*
PDST	*Path-Directed Subdivision Trees*
FMT	*Fast Marching Tree*
BFMT	*Bi-directional Fast Marching Tree*
lvsf	*longest_valid_segment_fraction*

Capítulo 1

Introdução

No primeiro capítulo desta dissertação é efetuada a contextualização ao leitor acerca da problemática da poda, bem como as motivações que levaram ao desenvolvimento de uma solução para a sua automatização. Posteriormente são delineados os objetivos da dissertação e é apresentada a sua estrutura.

1.1 Contexto

Os avanços tecnológicos nas últimas décadas têm permitido substituir a interação direta do ser humano em tarefas onde este outrora foi insubstituível. Além de melhorar a sua qualidade de vida, tais avanços permitem corresponder às necessidades de uma população mundial que se encontra em constante crescimento. Assim, a criação de sistemas e aplicações que reduzam a interação humana sem limitar a eficiência do processo e garantindo a mesma ou até melhor qualidade do produto final, é um fator crucial de sucesso no desenvolvimento de produtos no mercado atual. Dentro das vastas áreas de investigação, a robótica tem um impacto marcante pois desenvolve sistemas autónomos, independentes, robustos e eficientes.

A agricultura é uma das áreas que ainda muito depende de mão de obra. Desde o cultivo à colheita de vegetais ou frutos tem de haver uma monitorização e supervisão constante das culturas agrícolas que atualmente envolvem custos que nem sempre podem ser suportados pelos seus proprietários.

As vinhas são um dos ramos da agricultura que mais carece desta mão de obra devido a fatores como o terreno vinícola e sua dificuldade de acesso, a qualidade do fruto e ainda a poda das videiras. Estas são tarefas que envolvem capacidades físicas e cognitivas que atualmente são executadas pelo ser humano devido à sua capacidade de decisão e de auto suficiência.

Esta dissertação tem como foco a problemática da poda das videiras e estuda uma solução baseada em manipuladores robóticos que seja viável no mercado e alternativa às já desenvolvidas. Está inserida no projeto RoMoVi que tem como um dos objetivos desenvolver um robô para utilização em vinhas de encosta. Como sistema de testes será utilizado o AGROB V16, plataforma robótica de investigação e desenvolvimento do INESC TEC.

No subcapítulo 1.1.1 será feita uma breve introdução ao ciclo da uva, seguindo-se, no subcapítulo 1.1.2, uma abordagem mais detalhada à poda e à sua importância neste ciclo. A escrita destes subcapítulos está auxiliada não só nas referências deste documento como também no saber de Joaquim de Brito, viticultor há mais de 30 anos em Castelo Branco, e que partilhou o seu conhecimento nesta matéria.

1.1.1 Culturas vinícolas

Uma cultura vinícola é o nome pelo qual se designa a cultura de uvas. É uma das culturas de fruto de maior interesse em Portugal, devido à forte tradição na produção do vinhos, razão pela qual a sua qualidade é conhecida mundialmente. É, portanto, um tipo de cultura que requer uma intervenção constante, devido às diferentes fases do ciclo de produção de uva (ou ciclo do vinho), que exigem uma monitorização e supervisão constante de modo a garantir a qualidade do fruto e, consequentemente, a qualidade do vinho. Este ciclo dura cerca de um ano, começando entre Outubro e Novembro e encerrando em Agosto ou Setembro, altura em que é efetuada a vindima [1]. Sumariamente, o ciclo de vida desta cultura, em Portugal, resume-se aos seguintes momentos:

- Entre Outubro e Novembro, após a vindima do ano anterior, procede-se à lavra dos terrenos da cultura de modo a permitir uma maior infiltração das águas da chuva;
- Entre Novembro e Fevereiro, quando a planta se encontra no seu período de repouso vegetativo, isto é, quando não produz frutos, é realizada a poda lenhosa e a empa. A primeira operação consiste no corte de parte das suas vides, ao passo que a empa é o ato de dobrar e amarrar o pequeno tronco, resultante da poda das vides, conferindo assim estabilidade à planta, além de garantir que esta cresça com a forma pretendida;
- Entre Março e Abril começam a surgir as primeiras folhas e a formar-se os primeiros cachos da planta, pelo que é iniciado o ciclo de tratamentos para prevenir doenças e pragas;
- Entre Abril e Maio dá-se a floração da planta e é efetuada a poda em verde, isto é, são retirados à videira os rebentos acessórios, de modo a equilibrar a sua parte vegetativa (folhas) e produtiva (frutos);
- Entre Junho e Agosto os bagos desenvolvem-se e começam a adquirir a pigmentação característica e é iniciada a sua maturação;
- Finalmente, entre Agosto e Outubro é encerrado o ciclo com a vindima do fruto.

A poda ocorre em dois momentos do ciclo de vida de uma viticultura, podendo ser caraterizada de poda lenhosa, caso ocorra durante o período de repouso vegetativo, ou poda em verde, durante o período vegetativo. Pode ainda ser necessária noutros momentos deste ciclo de vida, e também durante os primeiros anos de vida da planta [2, 3]. O subcapítulo 1.1.2 detalha mais aprofundadamente a importância desta atividade.

1.1.2 A poda

A poda consiste no corte e remoção total ou parcial de certos órgãos da planta. Tem como objetivo regularizar o crescimento vegetativo e reprodutivo de acordo com um dado potencial de crescimento genético e ambiental de forma a assegurar a produção suficiente de uvas de qualidade e ainda a perenidade (longevidade) da planta [4]. Esta prática, nas videiras, consiste na remoção das vides, isto é, os rebentos que crescem nas suas ramificações principais, através de um corte efetuado segundo uma direção específica determinante para o sucesso desta prática. Ocorre em diferentes épocas do ciclo da uva, tendo diferentes objetivos consoante a finalidade segundo a qual é realizada. De acordo com [3], existem quatro tipos diferentes de poda: a poda de formação, a poda de manutenção ou frutificação, a poda de recuperação ou de rejuvenescimento e a poda de manutenção sanitária.

Poda de formação

Ocorre durante os primeiros três a cinco anos de vida da videira e tem como objetivo conferir à planta a estrutura de crescimento desejada, de forma a que esta se desenvolva segundo a ou as direções pretendidas. Além disso, deve garantir que os braços da planta estejam bem espaçados de forma a estarem facilmente acessíveis. A figura 1.1[1] exemplifica o resultado pretendido pela poda de formação ao longo dos 5 anos em que é efetuada.

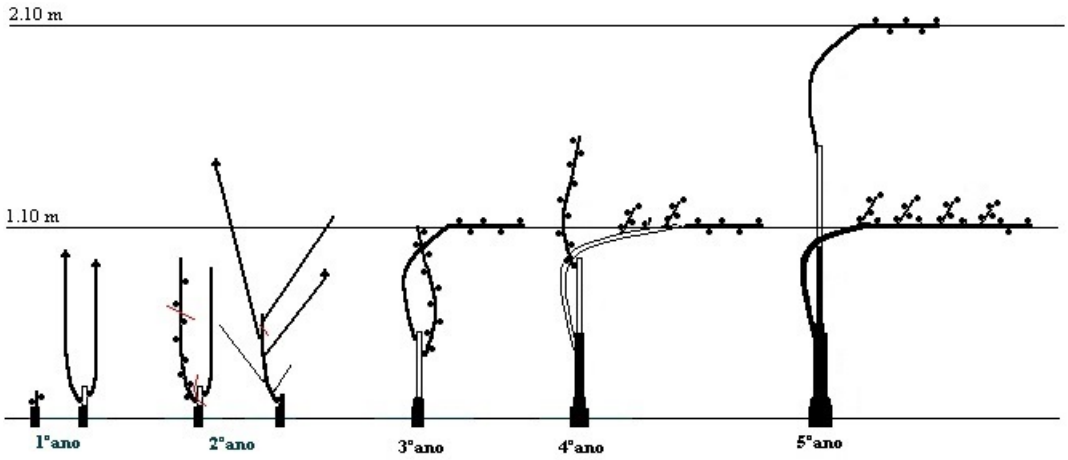


Figura 1.1: Efeito da poda de formação ao longo de cinco anos de vida de uma videira

Esta poda é a única que não se realiza durante o ciclo da uva, uma vez que nos primeiros anos de vida da videira a quantidade de fruto produzida é nula ou reduzida.

[1] http://portal.vinhoverde.pt/pt/file/c/1240

Poda de manutenção ou frutificação

A poda de manutenção ou frutificação é outra designação dada à poda lenhosa. É efetuada quando a planta já produz uma quantidade significativa de uvas e tem como objetivo a remoção de vides, segundo regras específicas de poda. Este tipo de poda realiza-se em geral no período de repouso, devendo ser executada quando não se verifique o aparecimento de novas rebentações. A figura 1.2[2] apresenta o exemplo do estado de um tronco de uma videira antes e após este tipo de poda.

(a) (b)

Figura 1.2: Tronco de uma videira a) antes da poda e b) após a poda

Poda de recuperação ou de rejuvenescimento

A poda de recuperação ou de rejuvenescimento é também conhecida como poda em verde. É semelhante à poda lenhosa mas ocorre numa fase mais avançada do ciclo, tendo como objetivo equilibrar a parte vegetativa e produtiva através da eliminação das vides "ladrões"que prejudicam o crescimento da planta. Estas vides são rebentos que não produzem fruto quando a maior parte dos restantes produz, desviando assim recursos necessários para o melhor desenvolvimento das vides de fruto. Por vezes são também cortadas as pontas destas vides de fruto para que não cresçam em demasia, melhorando assim o crescimento e maturação da uva.

Considere-se daqui em diante os ramos "ladrões"como vides, uma vez que, apesar de terem diferentes significados, o processo de corte em nada difere em ambos.

Poda de manutenção sanitária

A poda de manutenção sanitária pode ser necessária em qualquer idade da vida da videira. Consiste na eliminação de ramos doentes, danificados ou mortos, assim que aparecem, como forma de garantir que os mesmo não afetarão o desenvolvimento da restante planta. Pode também servir para eliminar ramos que estejam cruzados entre si, sobrepostos ou fracos e que estejam a impedir a circulação de ar e entrada de luz, essenciais para o desenvolvimento da videira.

[2] http://terramanhada.blogspot.com/p/vinha.html

O tempo de vida das videiras é normalmente superior a quarenta anos, pelo que a poda de formação é realizada durante uma pequena porção de tempo quando comparada com as restantes três, que se realizam durante todo o seu restante ciclo de vida. De forma resumida, a poda garante a:

- Formação e manutenção dos seus troncos;
- Regularização do potencial vegetativo, isto é, a quantidade de folhagem gerada nos seus troncos;
- Regularização da produção de uva (qualidade e quantidade);
- Correta distribuição dos ramos, que, entre outros aspetos, é determinante para a sua disposição em volta dos suportes às quais está presa durante o seu desenvolvimento.

A poda é portanto uma atividade essencial para a produção de uva. Caso não seja corretamente efetuada, interfere negativamente no crescimento da planta, assim como na qualidade e quantidade do seu fruto [5]. Este processo carece então de mão de obra especializada, pelas seguintes razões:

- A escolha das vides a serem removidas deve ser feita tendo em conta aspetos como o tipo de poda, que influencia a estrutura da planta e qualidade do fruto;
- As vides podem ser de difícil acesso, o que pode também dificultar a sua identificação;
- A remoção das vides rege-se através de regras que ditam a direção do seu corte;

1.2 Motivação

Atualmente a poda é realizada por agricultores que, através da sua experiência acumulada, sabem como e quando a efetuar. A presente dissertação pretende estudar e apresentar uma solução para a poda automática destes ramos, baseando-se num manipulador que estará montado na base do AGROB V16 (figura 1.3[3]), que irá percorrer as vinhas das encostas do Douro. A área de interesse abordada nesta dissertação é o planeamento de trajetórias a serem executadas pelo braço robótico no processo de corte das vides. Pretende-se que o estudo efetuado nesta dissertação seja um contributo importante na automatização da poda das vinhas.

1.3 Objetivos

O sistema de poda a integrar na plataforma AGROB V16 divide-se na componente de (1) perceção da vinha e de identificação das vides a podar e (2) numa outra para o planeamento de trajetórias. O trabalho a realizar consiste na segunda componente, sendo o que o principal objetivo é a avaliação do desempenho, ou, na designação anglo-saxónica, o *benhcmarking* de algoritmos

[3] http://agrob.inesctec.pt

Figura 1.3: Robô AGROB V16

de planeamento de trajetórias do manipulador, com vista à escolha da combinação e configuração que permita obter os mais baixos valores temporais para o planeamento de trajetórias, assim como a maior taxa de sucesso para trajetórias que não colidam com nenhum objeto no ambiente. A posição e orientação dos pontos de corte, *inputs* da primeira componente do sistema, tomam-se nesta dissertação como dados conhecidos *a priori*. O estudo será efetuado no braço robótico atualmente existente no AGROB V16, o *Robotis Manipulator-H*. A plataforma de desenvolvimento é o *Robot Operating System* (*ROS*), uma vez que os restantes módulos do AGROB V16 se encontram desenvolvidos nesta plataforma.

As principais tarefas definidas para o cumprimento deste objetivo são:

- Escolha de uma ferramenta de planeamento de trajetórias para integração com o braço robótico;
- Desenvolvimento do ambiente simulado onde serão efetuados os testes;
- Avaliação, em ambiente simulado, de algoritmos para o planeamento de trajetórias em aplicações de vinha;
- Avaliação, em ambiente real, de algoritmos para o planeamento de trajetórias em aplicações de vinha.

1.4 Estrutura da dissertação

Para além deste capítulo introdutório, a dissertação e composta por outros 5 capítulos. No capítulo 2 é efetuada a revisão bibliográfica, onde são apresentados os trabalhos anteriores a este, com foco na agricultura e na poda. Segue-se o capítulo 3 onde são apresentadas as ferramentas que permitiram a avaliação dos algoritmos para o planeamento de trajetórias. No capítulo 4 é abordado

o desenvolvimento do ambiente de simulação e a sua integração com a solução apresentada para o planeamento de trajetórias, seguindo-se no capítulo 5 a apresentação dos testes e respetivos resultados. Por último, o capítulo 6 com as conclusões, satisfação dos objetivos e trabalho futuro.

Capítulo 2

Revisão Bibliográfica

Neste capítulo são abordados os conceitos teóricos fundamentais ao entendimento e posterior desenvolvimento da dissertação.

Inicialmente é feita uma breve contextualização acerca do impacto da robótica na agricultura, sendo posteriormente efetuada a ponte para o desafio em questão e os projetos já desenvolvidos nesta área. Os conceitos abordados envolvem o estudo dos manipuladores para este tipo de aplicações, dos seus *end effectors* e dos algoritmos de planeamento de trajetórias.

2.1 Robótica na agricultura

A agricultura é um dos setores de mercado que desde sempre necessitou de mão de obra, pelo que a robótica surge neste ramo como uma via de automatização de todos os seus processos, pretendendo ser uma solução mais barata e viável. Destacam-se entre estes processos a plantação, a colheita e a poda [6], sendo que apenas a colheita e a poda são abordadas neste capítulo por serem ações que exigem movimentações semelhantes dos braços robóticos.

2.1.1 Robótica orientada para a colheita

Em culturas de grandes dimensões, a colheita exige elevada mão de obra, como é o caso das vindimas. Devido à sua importância, e sendo uma atividade comum a todas as atividades agrícolas, é um ramo onde já foram desenvolvidos bastantes projetos.

Bac *et al.* [7] aborda o estado da arte de robôs de colheita orientados para colheitas de alto valor, ou seja, que necessitam de intensiva mão de obra. Reviu 50 projetos dos últimos 30 anos relacionados com a colheita autónoma de frutos e vegetais usando sistemas que combinam visão computacional e braços robóticos. Identificou fatores de caraterização do ambiente de uma cultura de colheita e como estes podem influenciar o desempenho do robô que neles opera, dos quais se destacam os relacionados com as caraterísticas dos frutos ou vegetais da colheita, o ambiente em que esta se insere e o seu tipo.

Em relação às caraterísticas dos frutos ou vegetais, afirmam-se os seguintes fatores:

- Indicadores de maturação, tais como a cor do fruto ou vegetal, que influenciam a sua identificação;
- Mecanismos que permitam agarrar os elementos pretendidos, tendo estes diferentes tamanhos e formatos;
- A acessibilidade e visibilidade dos mesmos;
- A trajetória que deve ser efetuada pelo manipulador de modo a que este atinja o ponto pretendido sem danificar o restante ambiente;
- Tempo de vida do cultivo e práticas de cultivo dos agricultores (como o tipo de poda) que influenciam a geometria do fruto ou vegetal;
- Pestes e doenças que também influenciem o crescimento do fruto ou vegetal;

Em relação ao ambiente da colheita, destacam-se os seguintes fatores:

- Proteção de chuva e vento;
- Capacidade de controlar a luz incidente nos objetos;
- Condições climatéricas como temperatura, humidade e quantidade de CO_2 , entre outros;
- Visibilidade e acessibilidade dos objetos de interesse, condicionada pelo clima atmosférico;
- Capacidade de navegação no terreno.

Em relação tipo de cultivo, destacam-se os seguintes fatores:

- O objetivo para o qual se pretende o robô e que tipo de tarefas este terá que realizar no cultivo;
- O ciclo de vida do produto, que requer uma maior ou menor intervenção do robô;
- O mercado em que se insere.

No âmbito desta dissertação, existem vários fatores que podem ser extrapolados numa vertente orientada para a poda.

Nos fatores relacionados com as caraterísticas dos frutos e vegetais destaca-se a garra e demais mecanismos que lhe permitam cortar as vides. Estas vides comportam diferentes grossuras (que aqui podem ser entendidos como diâmetros) que influenciam a escolha da ferramenta, uma vez que esta é determinante na força a exercer pela ferramenta de corte.

Em relação ao ambiente de colheita, o principal fator a ter em conta é a proteção do manipulador contra chuvas e ventos, uma vez que a água pode danificar o seu *hardware* e a força do vento pode ter influência nos seus movimentos. O manipulador deve ser então dotado de um índice de

proteção adequado. As condições de visibilidade não são, nesta dissertação, um tema a abordar, uma vez que a deteção dos elementos de interesse não faz parte do seu âmbito.

Relativamente ao tipo de cultura, grande parte da discussão já foi abordada nos subcapítulos 1.1.1 e 1.1.2. O mercado da uva está intimamente ligado ao do vinho, havendo um consumo deste produto durante todo o ano. Assim, o ciclo da uva decorre anualmente, mantendo-se constante a produção de vinho, sendo a poda realizada em pelo menos duas instâncias deste ciclo, pelo que se prevê que o robô opere nessas duas épocas, podendo também efetuar os restantes tipos de poda abordados no subcapítulo 1.1.2. A inclinação do terreno é outro fator a ter em conta. As vinhas do Douro encontram-se na margens da sua encosta, pelo que o terreno é pedregoso e possivelmente inclinado, tendo isto impacto na cinemática do robô, e, consequentemente, na cinemática do manipulador, uma vez que a base onde está montado pode não se encontrar paralela ao terreno no momento em que este efetua a poda. Este problema não será tido em conta nesta dissertação, uma vez que as restrições por ele impostas agravam a dificuldade do problema, que por si só já se prevê complexo.

Ainda no estudo realizado por Bac *et al.* [7], são identificados resultados relativos aos projetos analisados, servindo como dados que permitem identificar o tipo de metodologias a adotar aquando do desenvolvimento de projetos orientados para a agricultura, bem como as vertentes dos mesmos que ainda carecem de desenvolvimento.

Para os 50 projetos realizados ao longo de 3 décadas, os seguintes valores foram aferidos relativamente às tarefas delegadas aos robôs:

- 66% de sucesso na colheita de frutos;
- 5% de frutos e vegetais colhidos danificados;
- 45% de caules da danificados;
- Tempo de ciclo de 33 segundos, que compreende a localização, identificação de maturação, recolha e transporte do fruto, e a movimentação do robô até ao próximo fruto.

Estes resultados revelam que, relativamente à colheita, os mecanismos usados para a recolha dos frutos ou vegetais ainda carecem de melhorias, face aos resultados esperados por um a pessoa: a percentagem de sucesso de recolha apresenta um valor baixo, dado que um operário consegue recolher praticamente qualquer fruto que encontre; a percentagem de elementos colhidos, ainda que apresente um valor baixo, certamente não é mais baixo do que o verificado quando uma pessoa realiza a mesma tarefa; a percentagem de caules danificados indica que aproximadamente metade destes troncos foi afetado devido à interação do robô, o que não contribui para a longevidade da vinha; o tempo de ciclo de recolha de cada elemento é muito alto quando comparado com o tempo despendido por um operário, que consegue realizar a mesma operação em menos de 5 a 10 segundos.

Nenhum dos sistemas usados nos projetos analisados foi comercializado, possivelmente devido aos resultados obtidos. Destaca-se ainda a falta de interação entre os agricultores e os investigadores, que certamente contribuiu dificultou a obtenção de melhores resultados nestes sistemas.

Relativamente às tecnologias usadas nos projetos, os resultados indicam que:

- A maior parte dos manipuladores usados foram desenhados com 2 a 7 graus de liberdade, sendo 3 o número mais comum;
- 74% das plataformas foram desenhadas para cada projeto específico;
- 78% dos braços robóticos foram desenhados para cada projeto específico;
- 98% dos *end effectors* foram desenhados para cada projeto específico.

Estes dados demonstram uma forte evidência para o facto de diferentes produtos e culturas requererem aplicações bastante específicas.

Considerações relativas aos braços robóticos para a poda de vinhas são abordadas mais à frente na secção 2.2.

2.1.2 Robótica orientada para a poda

A poda é uma atividade caraterística de alguns tipos de árvores. Cada planta tem o seu próprio tipo de poda, consoante os objetivos que se pretendem advir dela. A robótica orientada à poda das videiras apresenta já diferentes soluções, ainda que recentes, sendo a mais antiga datada de 2015.

A *Vision Robotics Corporation* [8] desenvolveu um sistema automático caraterizado por uma estrutura, puxada por um trator, que realiza a poda das videiras num ambiente controlado. No seu interior contém câmaras que permitem identificar as vides, que são removidas através de um corte efetuado por duas tesouras de poda que se encontram no extremo de dois manipuladores dispostos frente a frente. Estas tesouras são semelhantes aos modelos tradicionais frequentemente encontrados. Apesar de não serem fornecidos valores relativos aos resultados obtidos, é possível observar através do vídeo[1] da referência que o sistema funciona corretamente, ainda que por vezes as tesouras colidam de raspão com as vides.

Um sistema robótico para a poda de vinhas foi desenvolvido por Tom Botterill *et al.* [9, 10, 11]. Semelhante ao desenvolvido pela *Vision Robotics Corporation*, consiste numa estrutura que se move ao longo das linhas da vinha, mapeando tridimensionalmente cada videira, cujas vides, através de um algoritmo de inteligência artificial, são escolhidas para serem posteriormente podadas através de um manipulador. A estrutura contém no seu interior 3 câmaras, um gerador e um computador responsável por todo o controlo e automatização do processo. Contém ainda um braço robótico com 6 graus de liberdade, cujo *end effector* é uma broca conectada a um motor que a faz girar enquanto atravessa as vides. A estrutura é fechada de forma a garantir que as condições de visibilidade apenas estão limitadas às impostas pelos *LEDs* que se encontram no seu interior.

[1] https://www.visionrobotics.com/vr-grapevine-pruner

O sistema permite podar uma videira em 2 minutos, o mesmo tempo despendido quando a mesma operação é efetuada manualmente nesse tipo de videira. São podadas 8,4 vides por videira e o algoritmo usado para o planeamento de trajetórias é o *RRTConnect*.

Um sistema colaborativo entre o robô e uma pessoa foi desenvolvido por Bechar *et al.* em [12]. O sistema é composto por uma câmara a cores e um sensor de distância a laser que identificam a planta e as respetivas vides a podar. A imagem da planta e ramos são então disponibilizados ao utilizador através de uma interface que permite que este selecione quais quer cortar. Após esta seleção, o manipulador corta as vides, usando como *end effector* uma serra circular que gira à medida que o braço robótico se desloca.

A solução mais recente (2018) é um *Automated Guided Vehicle* (*AGV*) de pequenas dimensões chamado *Wall-Ye V.I.N.*, desenvolvido pela empresa francesa *Wall-Ye* [13]. É um robô orientado para vinhas de pequenas dimensões, que não cresçam além dos 80 cm de altura do robô. É portanto uma solução bastante limitada, mas robusta dentro do mercado em que se insere, que não é o do âmbito desta dissertação.

Apesar de já existirem várias soluções desenvolvidas, todas envolvem plataformas bastante complexas e que se destinam apenas ao terreno específico da zona para onde se concebe a plataforma. O AGROB V16 é um robô mais simples e que se pretende que permita realizar um maior número de tarefas, como são exemplos a colheita, rega, monitorização e poda. Além disso, relativamente aos 2 primeiros sistemas apresentados, não será possível controlar as condições de luminosidade, o que será um desafio para a primeira componente do sistema de poda, não abordada nesta dissertação. Por último, esta plataforma está a ser desenvolvida para um ambiente mais agressivo que o dos restantes projetos, uma vez que as encostas do Douro são bastante pedregosas e inclinadas. Existe por isso um grande potencial de mercado para esta plataforma, que se prevê adaptável para outro tipo de vinhas.

Os subcapítulos seguintes focam-se no estudo das caraterísticas dos manipuladores, na escolha da sua ferramenta de poda e ainda nos algoritmos de planeamento de trajetórias.

2.2 Manipuladores

Um manipulador ou braço robótico é um robô usado para manipular, que pode ser operado através de controlo remoto ou programado para operar autonomamente, como se pretende em aplicações robóticas. Tal como descrito em [14], um manipulador é constituído por uma base onde são conectadas em série ligações sólidas, através de juntas que contêm motores, sendo a sua última ligação denominada de *end-effector*, a ferramenta que realiza a operação para o qual o manipulador é pretendido. As suas juntas podem ser do tipo rotativas ou prismáticas, conferindo assim ao robô a cinemática e dinâmica específicas, que por sua vez ditam o seu volume de trabalho. A dinâmica estabelece a relação entre os movimentos do robô e as forças envolvidas, ao passo que a cinemática estabelece as relações entre os seus movimentos, desprezando as forças envolvidas,

isto é, dita as relações entre os ângulos e posições das suas juntas e a posição e orientação finais do *end effector* do manipulador.

A escolha do braço robótico adequado para uma determinada tarefa é complexa na medida em que tem que atender aos requisitos da aplicação em causa. Além disso, a escolha do seu *end effector* também exige um estudo prévio. Os seguintes subcapítulos abordam estes dois tópicos, analisando mais detalhadamente estas caraterísticas.

2.2.1 Caraterísticas dos manipuladores

Os manipuladores são robôs complexos pelo que as suas caraterísticas têm que ser tidas em conta consoante a aplicação para o qual este é pretendido. As mais relevantes são o seu *payload*, repetibilidade, velocidade, peso, alcance e índice de proteção [15]. Além destas, destacam-se ainda os seus graus de liberdade. Serão de seguida apresentadas e explicadas estas caraterísticas, assim como, quando possível, indicados valores das mesmas no âmbito desta dissertação.

O ***payload*** de um manipulador representa o peso máximo que este consegue suportar sem danificar a sua estrutura. No âmbito desta dissertação, o único peso que o manipulador têm que suportar é o do seu *end effector*, pelo que este aspeto é relevante pois a tesoura de poda apresentará certamente um peso superior ao *payload* do braço robótico.

A **repetibilidade** é uma medida que quantifica a precisão com que o *end effector* do manipulador se consegue posicionar num ponto e orientação desejados. É importante, uma vez que a maior parte das aplicações robóticas exigem elevada precisão de operação, como é o caso do posicionamento da tesoura na posição e orientação correta para o corte das vides.

A **velocidade** de operação do manipulador é definida pelas velocidades conjuntas das suas juntas, determinadas pelas velocidades máximas que os motores nelas instalados atingem. Esta caraterística é determinante para serem alcançados bons resultados temporais, pelo que é considerada na escolha do braço robótico. Ainda assim, nesta dissertação não é um parâmetro estudado pois apenas apenas se pretende encontrar uma solução para o planeamento de trajetórias e não a sua execução.

O **peso** do manipulador é uma caraterística importante para a sua assemblagem à estrutura à qual está preso. O seu valor varia consoante a tarefa para a qual foi desenhado, uma vez que fatores como o *payload* influenciam a sua robustez e dimensão. Em aplicações industriais, os manipuladores são montados em bases sólidas que suportam massas significativamente superiores às dos braços robóticos, pelo que o seu peso é até negligenciável quando comparado com as suas restantes caraterísticas. Nesta dissertação, o manipulador está montado na base da plataforma AGROB V16, que suporta o peso de uma pessoa adulta, pelo que também não será um fator determinante na sua escolha, dado que o peso de um manipulador para esta aplicação não supera o peso de uma pessoa.

O **alcance** do manipulador define a sua região de atuação, que deve ser suficiente para alcançar todos os pontos de interesse e, no caso das videiras, alcançar todos os pontos de corte das vides. O AGROB V16 posiciona-se de forma a que a sua estrutura esteja a cerca de 80cm de cada videira. O braço robótico está montado na base do AGROB V16 que se encontra a cerca de 80cm de altura, pelo que um alcance superior a 60cm é suficiente para atingir qualquer ponto da videira.

O **Índice de Proteção** (*IP rating*) classifica e qualifica o grau de proteção de equipamentos eletrónicos contra a intrusão de objetos sólidos e de água. É um índice definido pela norma internacional IEC 60529 e pela norma equivalente europeia EN 60529 [16]. Tem um caráter importante para esta aplicação, uma vez que o manipulador opera ao ar livre e está sujeito a condições ambientais agressivas, como poeiras ou chuva que podem danificar a estrutura ou até mesmo *hardware*, provocando um funcionamento deficiente ou avaria.

O número e tipo de juntas conferem **graus de liberdade** (*Degrees of freedom - DOF*) ao manipulador, ditando assim a sua cinemática e consequente região de operação. Diferentes configurações permitem abranger as mesmas regiões, mas um maior número de juntas implica uma maior complexidade na cinemática e dinâmica do robô, pelo que a escolha do manipulador deve tender em conta este *trade off*.

Existem ainda outras caraterísticas associadas ao manipulador, tal como o momento de inércia das suas juntas, a força máxima por estas suportada ou ainda o tipo de protocolo de comunicações implementado. No entanto estas caraterísticas não são relevantes para o âmbito desta dissertação.

Assim, as caraterísticas de interesse de um braço robótico, no âmbito desta dissertação são:

- Repetibilidade;
- Velocidade;
- Alcance;
- Graus de liberdade.

Com base nestes parâmetros, foram estudados alguns manipuladores que apresentam valores suficientes para satisfazer a operação pretendida.

2.2.2 Caraterização de manipuladores para o AGROB V16

Os uso de manipuladores é comum em aplicações industriais, pelo que este será o ponto de partida para o seu estudo. Os tipos mais comuns de manipuladores são o cartesiano, o cilíndrico, o polar, o *SCARA* e o de revolução, apresentados na figura 2.1[2] conjuntamente com o região onde operam.

[2] http://slideplayer.com/slide/5892913/

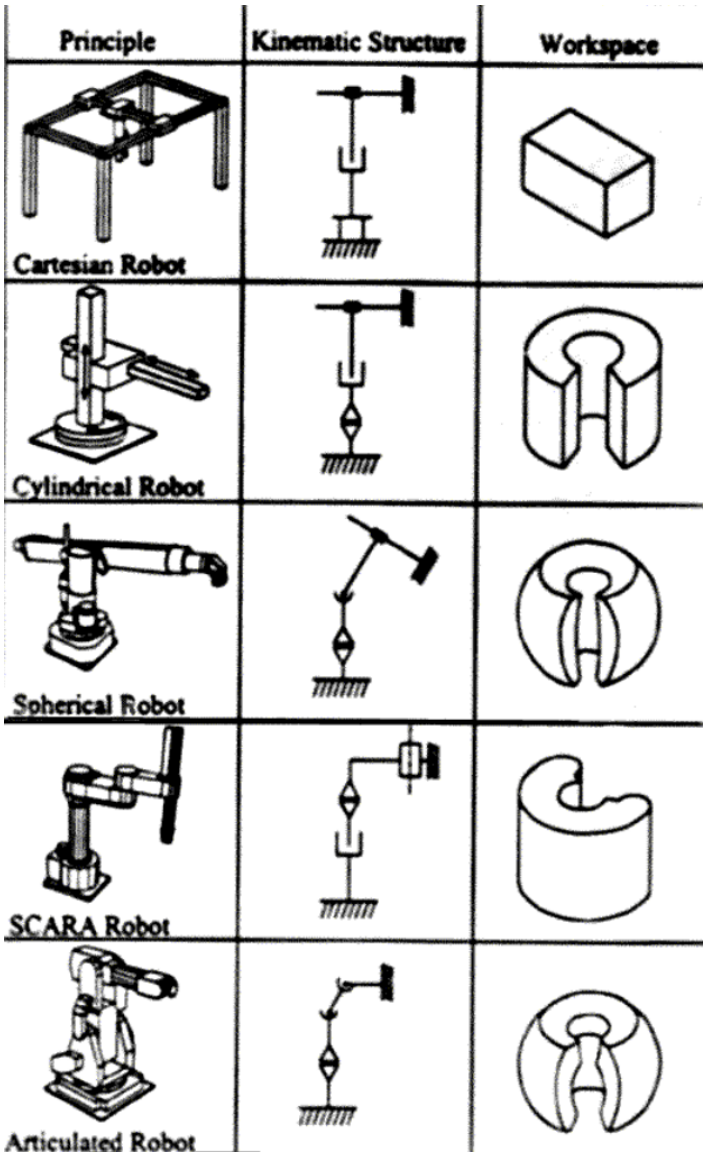


Figura 2.1: Diferentes tipos de manipuladores e respetiva região de operação

Estes manipuladores operam sobre meios controlados, onde os objetos a operar são estáticos e fáceis de detetar e as trajetórias do manipulador não estão restringidas por nenhum objeto no ambiente, daí que as suas configurações tenham predominantemente 3 graus de liberdade [14].

No entanto, o ambiente desfavorável da vinha dificulta o alcance das vides e dos seus restantes elementos, além de que as orientações dos pontos de corte requerem movimentos complexos do manipulador que muitas vezes só se tornam possíveis de atingir com vários graus de liberdade. Os manipuladores usados em [9, 12, 17, 11] têm 6 graus de liberdade, ao passo que os dois usados em [8], ainda que não seja explícito na referência, também aparentam ter os mesmos ou até mais graus de liberdade.

O âmbito da dissertação não cobre o desenvolvimento do manipulador, pelo que há a necessidade de integrar um na estrutura do AGROB V16 que permita atingir qualquer ponto de corte

desejado, mantendo a viabilidade económica da plataforma.

De momento, existe um manipulador montado na base do AGROB V16, resultado de estudos anteriores a esta dissertação, o *Robotis Manipulator-H* que se apresenta na figura 2.2[3]. As suas caraterísticas apresentam-se na tabela 2.1.

Figura 2.2: Manipulador *Robotis Manipulator-H*

Tabela 2.1: Caraterísticas *Robotis Manipulator-H*

Repetibilidade [mm]	± 0,05
Velocidade das juntas [º/s]	180
Alcance [mm]	645
DOF	6

Além deste braço robótico foram estudadas as soluções da bibliografia revista, que se encontram presentes no mercado e que podem ser soluções para esta problemática.

O sistema desenvolvido em [12] usa o manipulador *Motoman MH5LS II* (figura 2.3[4]), cujas caraterísticas se apresentam na tabela 2.2.

Figura 2.3: Manipulador *Motoman MH5LS II*

Tabela 2.2: Caraterísticas *Motoman MH5LS II*

Repetibilidade [mm]	± 0,03
Velocidade das juntas [º/s]	mín. 270
Alcance [mm]	895
DOF	6

[3] http://www.robotis.us/robotis-manipulator-h/
[4] https://www.motoman.com/industrial-robots/mh5ls-ii

Os sistemas desenvolvidos em [8, 9, 12, 11, 17] tomaram partido do braço robótico da *Universal Robotics*, *UR5* (figura 2.4[5]), cujas caraterísticas se apresentam na tabela 2.3.

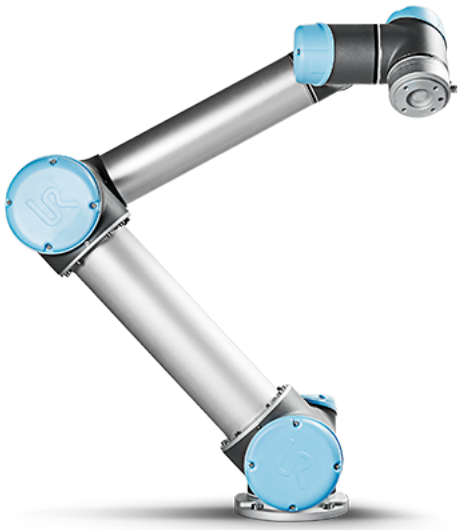


Figura 2.4: Manipulador *UR5*

Tabela 2.3: Caraterísticas *UR5*

Repetibilidade [mm]	± 0,1
Velocidade das juntas [º/s]	180
Alcance [mm]	850
DOF	6

A tabela 2.4 reúne as caraterísticas de todos os manipuladores, de modo a facilitar a sua comparação.

	Robotis Manipulator-H	*Motoman MH5LS II*	*UR5*
Repetibilidade [mm]	± 0,05	± 0,03	± 0,1
Velocidade das juntas [º/s]	180	mín. 270	180
Alcance	645	895	850
DOF	6	6	6

Tabela 2.4: Caraterísticas de todos os manipuladores abordados

Relativamente ao seu alcance, todos respeitam o valor mínimo indicado no subcapítulo 2.2.1, sendo que o *Motoman MH5LS II* é dotado do maior valor devido ao comprimento das suas partes metálicas. É também o manipulador mais rápido, uma vez que o valor mínimo para a velocidade atingida pelas suas juntas é de 270 º/s, sendo esta velocidade limitada a 180º/s para os dois restantes manipuladores. Quanto à repetibilidade, este manipulador é novamente o que apresenta o melhor valor, sendo portanto o preciso. No entanto, todos os braços apresentam valores considerados adequados para a aplicação em causa, uma vez que o comprimento das vides ronda 1,5m [3] e os valores das suas repetibilidades se encontram na ordem das décimas de milímetro, atingido valores que podem ser precisados por lâminas de corte.

Além disso, através da bibliografia revista verificou-se que os manipuladores usados em aplicações de poda eram de seis graus de liberdade, o que indica que esse número será condição suficiente para aplicações deste tipo.

[5] https://store.clearpathrobotics.com/products/universal-robots-ur5

Todos os manipuladores tem caraterísticas bastante semelhantes, sendo que o *Motoman MH5LS II* se destaca por apresentar os melhores. No entanto, o preço dos manipuladores é bastante elevado, pelo que a sua compra também é influenciada por este aspeto. O preço estimado [6] para o *UR5* é de 45,500$ [18], e como o preço do *Motoman MH5LS II* não deverá ser inferior (ainda que não se tenha conseguido obter uma estimativa fidedigna do seu valor), não se justifica o investimento nestes manipuladores. O preço do *Robotis Manipulator-H* é de $18,900 [19] pelo que é a opção mais viável para integrar o AGROB V16.

2.2.3 Caraterização de *end effectors*

O *end effector* de um manipulador é a ferramenta que se encontra conectada à sua ligação final e que lhe permite realizar a função para a qual é pretendido. Na indústria atual, são usados os seguintes tipos de *end effectors*: escovas, câmaras, ferramentas de corte, escavadoras, garras, ímanes, lixas, aparafusadores, pistolas de pulverização, aspiradores e pistolas de soldagem, entre outros [20].

A ferramenta usada para a poda manual de videiras é a tesoura bico de papagaio (*bypass pruner*), caraterizada pelas suas lâminas curvas que facilitam o corte das vides (figura 2.5[7]) [21, 22, 23].

Figura 2.5: Tesoura bico de papagaio (*bypass pruner*)

Os sistemas desenvolvidos em [8, 13] tomam partido de adaptações elétricas deste tipo de tesouras, oferecendo soluções com um tipo de corte bastante próximo do efetuado manualmente. O sistema realizado em [13] usa *Electrocoup F3015* (figura 2.6[8]), desenvolvida pela empresa francesa *Infaco*.

Esta tesoura tem um desenho muito similar às tesouras manuais, sendo no entanto elétrica, permitindo efetuar um corte automático através de um gatilho. A tesoura tem como vantagens o facto de conseguir aplicar uma força de corte superior à proporcionada por uma pessoa, sem alterar a orientação do corte, o que muitas vezes ocorre devido à força excessiva que é necessário aplicar na tesoura. Apresenta como principal desvantagem o facto do operador ter que carregar às costas

[6] Os preços por unidade variam consoante fatores tal como o número de equipamentos adquiridos aquando a sua compra, *etc*. Ver *websites* de algumas empresas.

[7] https://www.cutco.com/products/product.jsp?item=bypass-pruners#sm.0000xqrq3zek1f7dydm1dmzwq67kn

[8] https://www.infaco.com/pt/produtos/f3015/ficha-de-produto-f3015

Figura 2.6: Tesoura *Electrocoup F3015* desenvolvida pela empresa francesa *Infaco*

uma mochila que contém a bateria que alimenta a ferramenta. Apesar desta tesoura continuar ser operada por uma pessoa, a sua aplicação em [13] passou por uma adaptação do seu *hardware* para que pudesse ser operada automaticamente pelo robô através de sinais de controlo. Desta forma, o sistema movimenta o manipulador de forma a posicionar a garra na posição pretendida, sendo depois efetuado o corte. De notar ainda que esta é única tesoura elétrica certificada neste tipo de mercado [24].

O sistema desenvolvido em [9, 10, 11] utiliza um *end effector* semelhante a um aparafusador, que é o conjunto entre um motor e uma broca apresentado na figura 2.7.

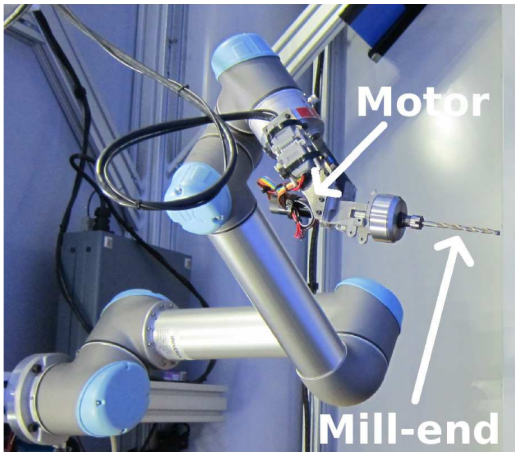


Figura 2.7: *End effector* constituído por um motor que faz rodar uma broca a alta velocidade [9]

Os cortes neste sistema são efetuados colocando a broca a girar a alta velocidade enquanto atravessa as vides. Esta solução é imprecisa, uma vez que, apesar de ser possível fazer com que a broca se movimente segundo a orientação de corte, mal esta entre em contacto com a vide a podar fará com que esta se movimente, o que pode condicionar alguns cortes. Um exemplo do corte efetuado com esta ferramenta apresenta-se na figura 2.8.

Figura 2.8: Corte efetuado pela broca numa vide [9]

Tal como se pode observar, o corte efetuado não é limpo, deixando algumas lascas e imprecisões indesejadas. Além deste problema, o sistema exige ainda que todos os movimentos de corte sejam efetuados em segmentos de reta de 10 cm, o que nem sempre é possível conseguir sem entrar em contacto com as outras vides ou ramos [9].

O sistema desenvolvido em [12] usa como *end effector* uma serra circular, tal como se apresenta na figura 2.9.

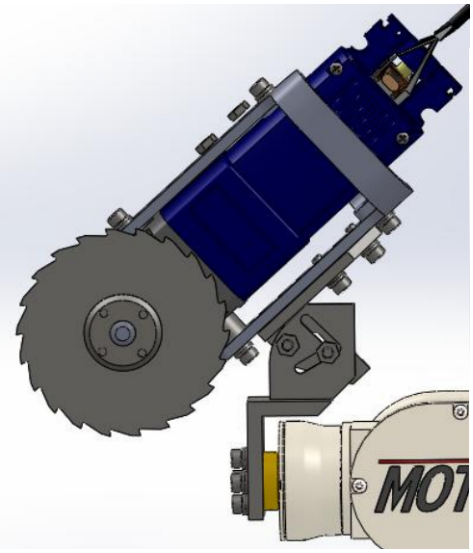


Figura 2.9: Serra circular usada em [12] como *end effector*

Tal como a ferramenta utilizada no sistema desenvolvido em [9, 10, 11], a serra circular não garante um corte limpo.

Uma vez que os *end effectors* que permitem cortes mais precisos para aplicações de poda são as adaptações das tesouras de poda, este tipo de ferramenta foi escolhido para o *end effector* do braço robótico *Robotis Manipulator-H*. Esta tesoura foi desenvolvida num projeto anterior a esta

dissertação, sendo no entanto as suas dimensões compatíveis com as do manipulador. A abertura máxima entre as lâminas na zona onde se pretende efetuar o corte com esta tesoura é de 1,5cm.

2.3 Algoritmos de planeamento de trajetórias do manipulador

Os movimentos de qualquer manipulador são complexos, dependendo não só da sua região de atuação como também da localização do objeto a atuar e do restante ambiente, que inviabilizam algumas trajetórias. O planeamento de trajetórias, na sua forma mais simples, entende-se como sendo o problema de investigação operacional que visa encontrar um caminho contínuo entre um estado inicial e um dado estado final, tendo em conta um dado conjunto de restrições [25]. Exemplos destas restrições são evitar colisões, manter uma dada orientação em certas partes do robô, limites de velocidade e de aceleração, entre outros.

A escolha de um método tem que tomar em consideração aspetos tais como:

- O tipo de otimização pretendida, tal como a distância da trajetória, o seu tempo de execução, o número de trajetórias corretamente calculadas ou a energia consumida na sua execução;
- A complexidade computacional, uma vez que a execução de alguns algoritmos é computacionalmente pesada, aumentando significativamente tempos de ciclo;
- A possibilidade de construir a solução baseando-se no modelo do ambiente antes de ser iniciada a trajetória (método *offline*), ou se a trajetória pode ser construída à medida que o robô se desloca (método *online*);
- Se existe um ou mais robôs, ou seja, se é necessário planear uma ou mais trajetórias;
- Se os obstáculos estão parados, em movimento previsível ou em movimento desconhecido (planeamento segundo restrições geométricas);
- Se o robô é holonômico ou não-holonômico, isto é, se se pode mover em qualquer direção ou se tem restrições de movimento;
- Se as restrições dinâmicas do robô entram no planeamento ou não (planeamento segundo restrições diferenciais);
- Se os obstáculos são deformáveis ou não;
- Se o método é completo, completo em resolução ou apenas probabilisticamente completo.

Um método diz-se completo quando encontra sempre uma solução, caso ela exista, e indica que não há solução, quando esta não existe. Um método completo em resolução é aquele que apresenta uma solução para uma determinada discretização do ambiente. Por fim, um método é probabilisticamente completo se a probabilidade de encontrar uma solução converge para 1 à medida que o tempo ou o número de amostras tende para infinito.

Todos estes aspetos tornam complexa a escolha do método. No entanto, o problema de planeamento de trajetórias pode ser efetuado segundo dois diferentes métodos: o planeamento segundo restrições geométricas (também conhecido por "*path planning*") e o planeamento segundo restrições diferenciais (onde também é abordado o "kinodynamic motion planning", isto é, onde são tidas em conta as restrições dinâmicas e cinemáticas do robô) [26].

Planeamento segundo restrições geométricas

Restrições geométricas são as impostas pelo ambiente de navegação, pelo que a mais comum é evitar colisões com obstáculos. Nelas, não há imposições efetuadas às velocidades ou binários do robôs, pelo que se supõe que este é capaz de realizar todo o tipo de movimentos. A este tipo de algoritmos de planeamento de trajetórias chamam-se planeadores geométricos (*geometric planners*) [27].

Planeamento segundo restrições diferenciais

Entendem-se restrições diferenciais com sendo aquelas associadas às derivadas temporais, que, teoricamente, existem em todos os sistemas. Têm origem em leis da física, no desenho do robô e nos requisitos das tarefas a serem desempenhadas por ele [28]. Um exemplo prático desta restrição é o facto de um carro não se poder deslocar a uma velocidade linear superior a 200km/h na direção perpendicular à do seu eixo de maior comprimento. Isto deve-se a vários fatores, tal como o facto da sua brecagem não permitir que as rodas do carro estejam orientadas segundo esta direção.

Os primeiros algoritmos desenvolvidos tomavam apenas em consideração restrições geométricas, como é o caso dos algoritmos *Bug* [29] que consistem em percorrer a linha que une o ponto inicial e final, contornando pela direita eventuais objetos que surjam na rota do robô. Além destes surgiram também os algoritmos de campos de potencial. Nesta abordagem, é usada uma função de potencial [30] que permite encontrar o melhor caminho entre o ponto inicial e o de destino, baseando-se no somatório de forças repulsivas e atrativas. A única força atrativa é alocada ao ponto de destino, sendo as forças repulsivas conferidas aos obstáculos ao longo do trajeto, pelo que é possível calcular o valor da função e, através da sua derivada temporal, obter-se a direção segundo a qual o robô se deve movimentar. Esta abordagem é bastante usada em aplicações de tempo real, uma vez que o robô "descobre"o caminho para o ponto de destino à medida que se movimenta, não sendo necessário um pré processamento para calcular o caminho mais curto antes do robô se movimentar. Tem como desvantagem o facto de poderem ocorrer pontos de mínimos locais, no caso em que o valor das forças repulsivas iguala o da força atrativa, fazendo com que o robô pare [31].

Os métodos atuais para a resolução de problemas de planeamento de movimentos conjugam restrições geométricas com restrições diferenciais, tomando partido dos mapas a explorar. Introduz-se aqui a noção de espaço de configuração, C_{space}, que representa o conjunto de todos

os pontos do robô, relativos a um referencial fixo no espaço euclidiano, e que é obtido através da translação do robô segundo os contornos dos obstáculos do mapa [32]. Assim, o espaço de configuração livre, C_{free}, representa todos os pontos onde é possível que o robô se posicione no mapa sem colidir com nenhum obstáculo, ao passo que o espaço de configuração dos obstáculos, C_{obs}, representa os pontos onde o robô não se pode posicionar. Todos estes pontos são dados por vetores de posição e respetiva orientação.

Uma vez que as técnicas de planeamento de movimentos se processam computacionalmente, há a necessidade de discretizar o C_{space}, pelo que surgem duas abordagens para este problema: planeamento de movimentos combinatório (*Combinatorial motion planning*) e planeamento de movimentos baseado em amostras (*Sampling-based motion planning*) [33]. Ambos se dividem em duas fases: a primeira, onde é construído um espaço de configuração livre que é mapeado num grafo; e a segunda, onde é efetuada uma pesquisa neste mesmo grafo pelo caminho mais curto. São abordados em maior detalhe nos seguintes subcapítulos.

2.3.1 *Combinatorial motion planning*

Combinatorial motion planning é o método de planeamento mais rígido, tanto na discretização do C_{space} como na procura do melhor caminho. Uma vez que os algoritmos deste tipo são completos, existe necessidade de mapear com maior exatidão o C_{space}. No entanto, esta rigidez torna-se computacionalmente pesada quando a dimensão do C_{space} aumenta [32]. Este método divide-se em duas diferentes topologias, consoante o tipo de C_{free} a ser construído: *roadmaps* e decomposição em células [34, 32].

Roadmaps

Consiste na procura do caminho mais curto numa rede de espaços amostrados chamada *roadmap*. O algoritmo pressupõe o conhecimento do mapa a percorrer, dividindo-se na fase de construção do *roadmap* e na fase de procura do seu caminho mais curto. Para a criação desta rede toma-se como ponto de partida o espaço de configuração do robô no mapa, C_{space}. O *roadmap* é então dado por todos os pontos centrais do robô que representam as posições do seu espaço de configuração livre, C_{free} (onde não há objetos intransponíveis), em que é possível colocar o robô, sendo construído através do teste do espaço de configuração do robô em posições aleatórias do mapa. Uma vez construído o *roadmap*, a procura do caminho mais curto resume-se a um problema de pesquisa em grafos. A solução para este problema é então o caminho mais curto que conecta todos os nós do *roadmap* desde o ponto inicial até ao ponto final, evitando colisões [35]. As diferentes técnicas de *roadmap* abordam diferentes métodos da sua construção bem como diferentes algoritmos de pesquisa em grafos. Destacam-se as técnicas de grafos de visibilidade (*visibility graphs*) [33] e os diagramas de *Voronoi* [36].

Decomposição em células

Esta técnica pressupõe também o conhecimento do mapa a percorrer. Analogamente aos *roadmaps*, este método também consiste na construção do mapa de células e na fase de pesquisa no mapa. O mapa de células consiste na decomposição do espaço livre do mapa (onde não existem obstáculos) em células, de tal forma que células adjacentes possam formar caminhos. De seguida, é construído um grafo que permite identificar as células adjacentes. A fase de pesquisa consiste novamente num problema de pesquisa em grafos, em que se pretende encontrar a cadeia de células que represente o caminho mais curto entre o ponto inicial e o final, sabendo que o caminho está livre de colisões, uma vez que as células apenas são criadas em espaços livre do mapa. Esta abordagem é também conhecida por *Boustrophedon Cellular Decomposition* [37]. As diferentes técnicas de decomposição em células abordam diferentes métodos de construção do mapa de células bem como diferentes algoritmos de pesquisa em grafos. Destacam-se as técnicas de decomposição do mapa em células exatas e a decomposição do mapa em células aproximadas [38, 32].

Existem ainda outras abordagens, tais como a programação matemática e as redes neuronais, não tão desenvolvidas nem aplicadas como as anteriores.

Na procura de soluções alternativas e mais eficientes para esta problemática, surgem os algoritmos baseados em amostragem, propostos de modo a superar a complexidade do planeamento de trajetórias para um robô com 6 graus de liberdade [39], como é o caso dos manipuladores analisados no subcapítulo 2.2.2.

2.3.2 *Sample-based motion planning*

Sample-based motion planning é um método de planeamento probabilisticamente completo, pelo que se torna torna mais eficiente que o primeiro para sistemas de altas dimensões [32]. Nesta abordagem, a mostragem do C_{space} é mais displicente, garantindo na mesma um bom compromisso entre os estados que vão sendo amostrados e a procura do melhor caminho nestes.

Nesta abordagem foram desenvolvidas três categorias de algoritmos: *roadmaps* probabilísticos, algoritmos baseados em árvores e outros métodos de decomposição em células [27].

- *Roadmaps* probabilísticos

Analogamente ao método *roadmap* descrito no subcapítulo 2.3.1, este também se divide numa fase de criação do *roadmap* e na fase de pesquisa pelo caminho mais curto. Inicialmente são amostrados estados aleatórios do C_{space} durante um certo período de tempo limitado. De seguida, são eliminadas todas as conexões entre estados onde não é possível estabelecer uma linha de visão. Finalmente, com os restantes estados amostrados é construído o *roadmap*, sendo usado como critério de conectividade entre estados todas as amostras que se encontram dentro de um dado raio ou ainda um certo número de vizinhos mais próximos. Por último, é nele efetuada uma pesquisa que procura o caminho mais curto [27, 32];

- Algoritmos baseados em árvores

Consiste em explorar de forma abrupta o C_{space}, partindo de um estado inicial aleatório. A árvore é iterativamente construída através de estados que são aleatoriamente amostrados e que são posteriormente conectados àqueles que se encontram mais próximos dos nós da árvore. As ligações formadas entre estados são inicialmente de grande comprimento, sendo que este vai diminuindo à medida que a árvore vai sendo construída. Desta forma, ao fim de algumas iterações, é formado um caminho que une o estado inicial ao estado de destino [27, 32];

- Decomposição em células

Este método difere em grande parte dos dois anteriores. Linhas verticais vão sendo projetadas no C_{space}, e quando uma destas linhas interseta um canto de um obstáculo ou do mapa, uma célula de ligação é criada. Quando o C_{space} for totalmente analisado, a criação das células é cessada e estas são conectadas segundo critérios a definir.

Todos os sistemas analisados onde é efetuada uma comparação entre algoritmos referem apenas algoritmos estado da arte do tipo *Sample-based motion planning*, demonstrando o seu impacto no panorama atual da problemática do planeamento de trajetórias.

O seguinte subcapítulo apresenta resultados de comparações efetuadas em sistemas onde o planeamento é feito em duas e três dimensões. São apresentados algoritmos que ainda não foram referidos mas que serão mais tarde apresentados no subcapítulo 3.2, uma vez que estão incluídos na *Open Motion Planning Library (OMPL)*.

2.3.3 Comparação de algoritmos

Dados recolhidos de anteriores sistemas permitem retirar conclusões acerca das caraterísticas de cada algoritmo e dos resultados que se podem esperar da sua aplicação.

Em [27] é feita uma analise comparativa entre vários algoritmos para resolver o problema de encontrar um caminho entre um ponto inicial e um final no mapa simulado da figura 2.10, que é percorrido por um carro. Este é um mapa bidimensional, pelo que também o são as dimensões do problema a resolver.

Numa primeira abordagem a solução é implementada usando planeadores geométricos, sendo comparadas as versões geométricas dos algoritmos *RRT*, *PRM*, *EST* e *KPIECE*. De seguida, a solução é implementada usando planeadores baseados em controlo, sendo comparadas estas versões dos algoritmos *RRT*, *PRM* e *EST* (de notar que aqui entra também em conta o modelo dinâmico do carro). Finalmente, são comparados nas suas versões geométricas os algoritmos *RRT* e *PRM* com as suas versões ótimas, *RRT** e *PRM**, e ainda é feita uma comparação entre estes dois últimos.

Relativamente à primeira abordagem, *KPIECE* e *EST* apresentam resultados temporais significativamente piores que *RRT* e *PRM*. A escolha entre este dois últimos depende do critério a otimizar: caso o mais importante seja encontrar o caminho mais curto, então o *PRM* é um método

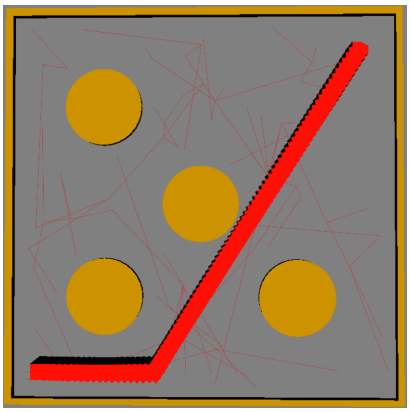


Figura 2.10: Mapa desenhado em [27] para a comparação de desempenho entre algoritmos (o traçado vermelho é o exemplo de uma solução)

mais efetivo, apesar de tomar mais tempo de processamento; caso seja necessário otimizar este tempo, então a melhor solução é o *RRT*.

Para os métodos baseados em controlo, o *KPIECE* surge como a melhor solução uma vez que apresenta o menor valor da função de custo, apesar de todos os métodos despenderem do mesmo tempo para o planeamento.

Relativamente à comparação entre *RRT* e *RRT**, a conclusão a que se chega é que o primeiro atinge uma solução cerca de 100 vezes mais rapidamente que o segundo, ao passo que este último é mais preciso, uma vez que é ótimo e por isso garante necessariamente o caminho mais curto.

Comparando o *PRM* com a sua versão ótima, o *PRM**, o tempo de execução e encontrar o caminho ótimo são os dois fatores a ter em conta. Caso se pretenda uma solução rápida em detrimento de um caminho ótimo, então a escolha é o *PRM*, ao passo que a situação contrária requer o algoritmo *PRM**.

Por último, a comparação entre o *RRT** e o *PRM** revela que o primeiro tem um tempo de execução mais rápido, pelo que é claramente melhor que o segundo.

Em [40] são abordados os algoritmos *RRT* e *KPIECE* da *OMPL* para o estudo do planeamento de trajetórias em manipuladores industriais com 6 graus de liberdade. Conclui-se que os planeadores baseados em *RRT* produzem resultados mais precisos e rápidos que os baseados em *KPIECE*.

Os sistemas desenvolvidos em [41, 10] abordam o problema da poda, sendo o robô deste sistema novamente um manipulador com 6 graus de liberdade. Usam como algoritmo de planeamento o *RRTConnect** (versão bidirecional do algoritmo *RRT**, não contido na *OMPL*), sujeito a restrições geométricas e ainda uma versão evoluída do algoritmo, construída para melhorar a problema específico deste sistema, e que produz melhores resultados temporais que a versão original.

Após revisão da bibliografia citada, conclui-se que os algoritmos da família *RRT* apresentaram resultados melhores, em termos temporais, que os seus concorrentes, no que toca a aplicações de

planeamento de trajetórias bidimensional e tridimensional. Além disso, as aplicações práticas revistas tomam partido deste algoritmo no planeamento de trajetórias de manipuladores com 6 graus de liberdade, como é o caso de todos os abordados no subcapítulo 2.2.2. Assim, a escolha deste algoritmo para a problemática específica da poda em [41, 10], revela uma forte evidência para o seu uso no decorrer desta dissertação.

Além destes algoritmos, existem outros do tipo *Sample-based motion planning*. Como forma de integrar todas as suas implementações numa só ferramenta, Ioan A. *et al* desenvolveram em 2012 uma biblioteca que reúne os mais importantes algoritmos deste tipo, a *OMPL*, já anteriormente referida. Como todas as soluções encontradas remetem para algoritmos implementados nesta biblioteca, esta foi tomada como o ponto de partida para encontrar uma solução para a problemática desta dissertação. A *OMPL* e seus algoritmos são abordados em maior pormenor no subcapítulo 3.2.

No próximo capítulo é explicada a abordagem concreta que foi tomada relativamente à plataforma onde foi desenvolvida a dissertação, assim como o *software* utilizado que permitiu tomar partido dos algoritmos implementados na *OMPL*.

Capítulo 3

Ferramentas de desenvolvimento

Neste capítulo é apresentado o ambiente de desenvolvimento apresentado nesta dissertação. Inicialmente é feita uma breve introdução ao *ROS*, sendo de seguida apresentados os seus *plugins*: os simuladores *Gazebo* e *RViz* e o programa *MoveIt!* para o planeamento de trajetórias. É de seguida feita a ponte entre o *MoveIt!* e a *OMPL* e por último apresentados os planeadores. De notar que nem todos são apresentados com igual pormenor, devido ao facto da quantidade de artigos publicados não ser uniforme.

3.1 *ROS*

No panorama atual do desenvolvimento de aplicações, a componente de simulação tem um papel fundamental, permitindo testar o desenvolvimento de produtos, assim como detetar possíveis falhas que comprometam o seu correto funcionamento e viabilidade.

Empresas com grande afirmação no mercado desenvolvem os seus próprios *softwares* de simulação (como é o caso da *ABB* e da *Fanuc*), diferenciando-se assim dos demais concorrentes. No entanto, esta dissertação está a ser desenvolvida utilizando robôs de diferentes marcas, pelo que a sua plataforma de desenvolvimento tem que suportar esta diversidade.

Surge assim o *Robot Operating System* (*ROS*), uma *framework* flexível para o desenvolvimento de *software* robótico que, através de uma vasta coleção de bibliotecas, ferramentas e convenções, consegue simplificar a criação de aplicações robóticas que integrem as demais plataformas robóticas do mercado. Mais, sendo *open-source*, o *ROS* permite a integração entre o trabalho desenvolvido por diferentes companhias ou laboratórios, mantendo um crescimento constante da sua comunidade e dos conteúdos que lhe são compatíveis. Por exemplo, um laboratório pode ser especializado no mapeamento de ambientes *indoor* enquanto que outro laboratório pode estar especializado em visão computacional. Através da publicação dos seus trabalhos para a comunidade *ROS*, estes podem servir de base para outros que sejam posteriormente desenvolvidos.

O *ROS*, por si só, não é um programa, ambiente de programação ou de simulação. Em vez disso, ele assenta no conceito de nós, que representam uma ou mais partes do robô, como sensores e atuadores. Estes nós comunicam entre si através da subscrição e publicação de mensagens

(específicas do *ROS*) em tópicos. Um exemplo da interação entre nós através de tópicos pode ser visto na imagem 3.1.

Figura 3.1: Exemplo da interação em *ROS* entre 2 nós através de um tópico

A integração de vários sistemas consiste apenas no adição de mais nós e do estabelecimento de comunicações entre eles através dos ditos tópicos. O código nesta *framework* encontra-se organizado dentro de *packages* que, tal como o nome indica, são pacotes que contém vários ficheiros de código que por usa vez contêm funções. Para que o utilizador possa tomar partido delas, estas funções estão associadas a nós. O lançamento (*launch*) dos nós pode ser efetuado através da linha de comandos, sendo que, no momento em que ganham "vida", os nós automaticamente publicam e subscrevem tópicos específicos (ainda que novas subscrições e publicações possam ser despoletadas mais tarde).

É possível assim perceber a escalabilidade do *ROS* e a dinâmica que o seu funcionamento impõe. Além disso, o seu conceito permite dividir claramente as camadas de código inerentes ao *ROS* daquelas que lhe são independentes, permitindo a compatibilidade com outros sistemas.

Há ainda diferentes versões do *ROS*, que foram desenvolvidas de forma a otimizar o seu funcionamento. Atualmente, a versão que se encontra suportada por mais tempo pela sua equipa é o *ROS Kinetic*, suportado até 2021. Esta é a versão utilizada nesta dissertação.

Existem atualmente alguns pacotes do *ROS* que podem ser entendidos como os seus *plugins* essenciais. Tratam-se daqueles que lhe permitem, além de simular o robô, programar ações relacionadas com visão ou até o movimento de manipuladores. Os mais importantes e usados nesta dissertação são apresentados de seguida.

3.1.1 *Gazebo*

Figura 3.2: Logótipo do *Gazebo*

Gazebo é um simulador tridimensional de ambientes interiores e exteriores para múltiplos robôs, completo com físicas de dinâmica e cinemática para melhor representar um ambiente real. Ainda que independente do *ROS*, é possível a integração entre ambos através de um conjunto de

packages que suportam vários robôs, sensores e atuadores. Estes utilizam o mesmo tipo de mensagens do ecossistema *ROS*, pelo que as aplicações desenvolvidas em simulação podem ser aplicadas em robôs físicos com poucas ou nenhumas modificações no código, desde que a realidade esteja devidamente simulada.

Este simulador serve assim como ponto de partida para a análise a ser efetuada para a plataforma para a poda da vinha. Existem *online* vários robôs desenvolvidos para operarem nesta plataforma, assim como ambientes de simulação, sensores e atuadores e pacotes que operam com eles de forma a garantir o intercâmbio de dados, tal como aconteceria num ambiente real. Ainda assim, um robô específico como o AGROB V16 não se encontra disponível *online*, uma vez que faz parte de um projeto confidencial e porque integra robôs de vários tipos (um *AGV* e um braço robótico, de momento) e ainda de diversas marcas. Parte do trabalho desta dissertação passa então por desenvolver a correta representação desta plataforma para este simulador.

Em *ROS*, um robô é descrito, por convenção, no formato *Unified Robot Description Format* (*URDF*). Programado em linguagem *XML* e em *Xacro* (*XML Macros*), esta linguagem é orientada para esta aplicação pois assenta no conceito de *tags*, que neste caso representam as componentes e ligações entre elas no robô, assim como as propriedades que lhes são inerentes, tal como a sua posição inicial, rotação inicial, entre outras.

Uma vez que o simulador *Gazebo* é independente do *ROS*, também apresenta o seu próprio formato descritivo para robôs, o *Simulation Description Format* (*SDF*), bastante semelhante ao *URDF* mas mais completo. Como a integração entre *ROS* e o *Gazebo* já se encontra completamente estabelecida, o *Gazebo* consegue automaticamente converter ficheiros *URDF* em *SDF* pelo que, para implementar corretamente um robô na simulação, apenas é necessário configurar os seus ficheiros *URDF*.

3.1.2 *RViz*

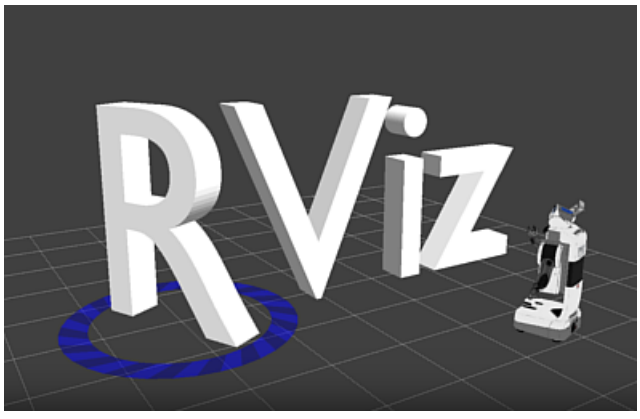


Figura 3.3: Logótipo do *RViz*

O *RViz* é um simulador desenvolvido especificamente para o ambiente *ROS* que permite visualizar os dados provenientes dos sensores simulados. É uma ferramenta que complementa as simulações em ambiente *Gazebo*, uma vez que permite visualizar aquilo que o robô realmente

perceciona do ambiente onde está inserido, ao passo que o *Gazebo* permite simular esse mesmo ambiente. Além de informações dos sensores do robô, permite ainda desenhar qualquer tipo de marcadores, visualizar trajetórias, *etc.*. Nesta dissertação tem um papel importante, uma vez que o *MoveIt!* opera neste *software* de forma a tornar possível planear e visualizar as trajetórias do manipulador num *Graphical User Interface* (*GUI*) inserido numa janela do *RViz*.

3.1.3 *MoveIt!*

Figura 3.4: Logótipo do *MoveIt!*

O *MoveIt!* é uma biblioteca que oferece implementações eficientes de algoritmos estado de arte de planeamento de trajetórias. É um pacote compatível com todo o tipo de robôs, desde plataformas semelhantes a carros até manipuladores e robôs humanoides. Este *plugin* pode ser usado por qualquer robô suportado pelo *ROS* e que esteja descrito no formato *URDF*, condição indispensável para as configurações do *MoveIt!*.

Esta aplicação é bastante recente, tendo surgido em 2013 e estando em constante expansão e desenvolvimento, com a integração constante de novas funcionalidades. Atualmente, permite:

- Configurar o robô a ser controlado através do seu próprio ambiente de simulação;
- Efetuar o planeamento de trajetórias através do *GUI* presente no simulador *RViz*;
- Programar as trajetórias através de código e visualizar e desenhar no *GUI* do *RViz* marcadores que representem, por exemplo, os pontos inicial e final das trajetórias.

A grande vantagem do *MoveIt!* é o facto de conseguir efetuar o planeamento de trajetórias tendo em conta objetos intransponíveis do ambiente, desde que sejam por ele detetados, permitindo que o cálculo das trajetórias seja efetuado tendo em conta estes obstáculos. Desta forma são evitadas colisões do robô com o meio ambiente, condição indispensável para a aplicação pretendida, devido à heterogeneidade da disposição das vides nas vinhas. A explicação detalhada de todo o processo que envolve o *MoveIt!* na dissertação será abordada mais à frente no subcapítulo 4.2.

O *MoveIt!* foi desenhado para operar com uma vasta gama de planeadores, o que facilita o seu *benchmarking* para aplicações específicas, como é o caso. Atualmente existem 4 bibliotecas de planeadores implementadas no *MoveIt!*, de seguida apresentadas por ordem descendente de popularidade e suporte no universo desta aplicação.

Open Motion Planning Library (*OMPL*)

A *OMPL* é uma biblioteca de planeamento de trajetórias *open-source* que implementa uma vasta gama de planeadores estado de arte do tipo *Sample-based motion planning*. Os planeadores da *OMPL* são abstratos, isto é, não têm na sua definição e implementação da *OMPL* nenhum conceito de robô. O *MoveIt!* complementa assim esta definição através da configuração da *OMPL* com o *back-end* necessário para que esta opere em problemas de robótica. Esta biblioteca é completamente suportada pelo *MoveIt!*.

Stochastic Trajectory Optimization for Motion Planning (STOMP)

STOMP é a implementação de um planeador baseado no algoritmo *Policy Improvement with Path Integrals* (PI^2) [42]. Planeia trajetórias suaves para braços robóticos, evitando obstáculos e otimizando restrições. Encontra-se, de momento, parcialmente suportada.

Search-Based Planning Library (SBPL)

Esta biblioteca reúne um conjunto de planeadores que se baseiam num método de pesquisa que discretiza o espaço. A integração desta biblioteca no *MoveIt!* encontra-se neste momento em desenvolvimento.

Covariant Hamiltonian Optimization for Motion Planning (CHOMP)

CHOMP é um algoritmo de otimização de trajetórias baseado em gradientes que torna vários movimentos do dia a dia tanto simples como treináveis [43]. Enquanto que planeadores multi dimensionais separam a geração de trajetórias na fase de planeamento e na fase de otimização, este algoritmo usas as abordagens do gradiente covariante e funcional para a fase de otimização, de forma a desenhar um algoritmo de planeamento de trajetórias baseado inteiramente na otimização da trajetória. A integração desta biblioteca no MoveIt! encontra-se neste momento em desenvolvimento.

De todas as bibliotecas do *MoveIt!*, apenas a *OMPL* se encontra totalmente suportada e implementada, sendo por isso a que vem por defeito nas suas configurações. Além disso, e tal como já referido, esta biblioteca implementa os algoritmos analisados no capítulo 2. Por estas razões, apenas esta biblioteca e seus algoritmos são objeto de estudo nesta dissertação.

No próximo subcapítulo é efetuada uma introdução a todos os algoritmos da *OMPL* que são implementados pelo *MoveIt!*.

3.2 *Open Motion Planning Library*

3.2.1 Introdução

A *Open Motion Planning Library*[1] é uma biblioteca desenvolvida com o intuito de ser um ponto de partida para o desenvolvimento de aplicações que necessitem da implementação de algoritmos de planeamento de trajetórias bidimensionais e tridimensionais, como é o caso dos sistemas robóticos [44]. Contém implementações de vários algoritmos do tipo *Sample-based motion planning*, alguns dos quais foram já citados no subcapítulo 2.3.3.

Na *OMPL* existem duas categorias de planeadores de movimento, os geométricos (*Geometric Planners*) e os baseados em controlo (*Control-based planners*). Os primeiros tomam apenas em conta as restrições geométricas e cinemáticas do robô e assumem que qualquer caminho viável pode ser transformado num trajetória passível de ser executada [44]. Os planeadores baseados em controlo partem do princípio que o sistema considerado está sujeito a restrições diferenciais, como por exemplo, as velocidades e acelerações máximas que o robô pode atingir. Assim, uma solução robusta deve ter em conta restrições de ambos os tipos. Sendo os planeadores geométricos mais fáceis de conceber, uma vez que são mais genéricos que os baseados em controlo, a maior parte dos algoritmos surgiram inicialmente como geométricos, sendo mais tarde adaptados para uma versão baseada em controlo (à exceção do *KPIECE*, cuja situação foi inversa) [27].

No *MoveIt!* apenas estão implementados alguns planeadores geométricos da vasta biblioteca que é a *OMPL*, pelo que o foco deste capítulo são os planeadores deste tipo. O principal objetivo da integração da *OMPL* no *MoveIt!* é que este calcule, tanto em aplicações bidimensionais como tridimensionais, uma trajetória livre de colisões que conecte as posições inicial e final do *end effector* do manipulador com o menor número de pontos possível, sendo esta trajetória um conjunto de coordenadas cartesianas (x, y, z) sequenciais que denotam os pontos do espaço a ser percorrido.

Existem dois tipos de planeadores geométricos, os *single-query* e os *multi-query*. Os primeiros são aqueles que executam as tarefas de criação do *roadmap* e de pesquisa do caminho sequencialmente, isto é, uma seguida da outra. Nos segundos, ambas as fases são executadas simultaneamente, limitando ou adaptando o tamanho do *roadmap* durante a fase de pesquisa. Nos seguintes subcapítulos são apresentados os diferentes algoritmos deste tipo, sendo primeiro expostos os algoritmos *multi-query* e em seguida os *single-query*.

3.2.2 Planeadores *multi-query*

Os algoritmos dos planeadores *multi-query* baseiam-se na construção de um *roadmap* de todo o mapa, onde várias *queries* de pesquisa vão sendo executadas ao longo de todo o processo de construção do mapa. São de seguida apresentados os algoritmos deste tipo implementados no *MoveIt!*.

[1] http://ompl.kavrakilab.org

Probabilistic RoadMap (*PRM*)

PRM é um planeador capaz de calcular caminhos sem colisões para todo o tipo robôs que se movam em ambientes com obstáculos fixos [27]. Consiste na execução de duas fases síncronas: a construção do *roadmap* e a procura do seu melhor caminho. O mapa é construído através da seleção aleatória de uma configuração do robô, sendo investigado se existe ou não uma colisão do robô com algum obstáculo. Quando uma configuração livre é encontrada, esta é conectada aos restantes nós do *roadmap* segundo critérios a definir (como por exemplo, os primeiros k nós que se encontrem a uma distância menor que um valor predefinido). A figura 3.5[2] apresenta um exemplo da construção do *roadmap* neste algoritmo.

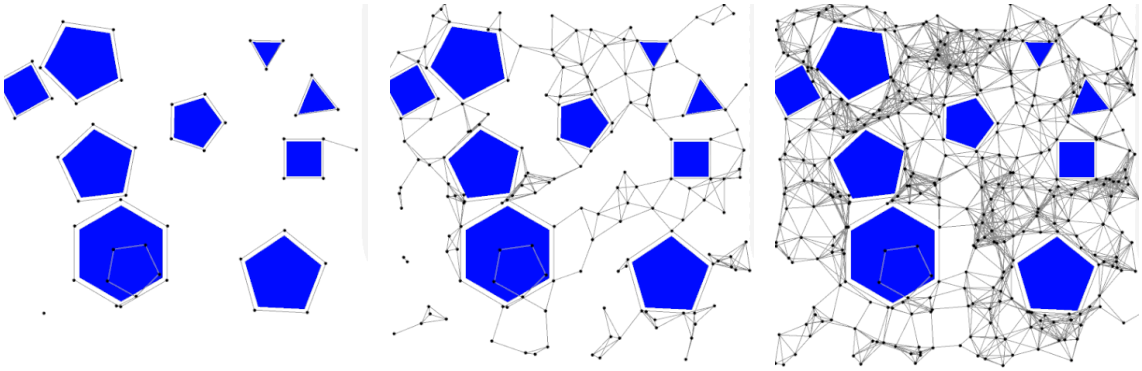


Figura 3.5: Exemplo de sucessivas iterações da construção do *roadmap* segundo o algoritmo *PRM*

A pesquisa pelo caminho mais curto é constantemente efetuada por uma tarefa que atua paralelamente à que constrói o *roadmap*, sendo que o método implementado para a pesquisa é o algoritmo $A*$ (o algoritmo de *Dijkstra* também é frequentemente utilizado nesta fase).

Lazy Probabilistic RoadMap (*LazyRPM*)

O algoritmo *LazyPRM* é uma vertente do algoritmo original *PRM* que tem como principal objetivo minimizar a verificação de colisões entre o robô e o C_{space} e, consequentemente, minimizar o esforço computacional despendido no planeamento da trajetória.

O algoritmo constrói um *roadmap* do C_{space}, fazendo com que nós vizinhos sejam conectados através de vértices de modo a representar os caminhos entre eles. Contrariamente à implementação original do algoritmo, este planeador assume inicialmente que todos os nós e vértices do *roadmap* não colidem entre si, verificando entre todos os nós qual é o melhor caminho entre o nó inicial e o final. Uma vez encontrado o melhor caminho, todos os seus nós são verificados quanto a pertencerem ao C_{free} ou C_{obs}, daí o algoritmo ser preguiçoso, pois apenas verifica a validade do caminho após o encontrar. Todos os nós que pertençam ao C_{obs} são então eliminados do C_{space} de forma a facilitar futuras pesquisas.

O algoritmo permite que, após cada iteração de pesquisa pelo caminho mais curto, ocorra uma de duas situações: ou é encontrado um novo caminho mais curto entre o estado inicial ou final,

[2]Fonte: https://en.wikipedia.org/wiki/Probabilistic_roadmap

ou o *roadmap* é atualizado, tornando mais eficientes as subsequentes pesquisas que nele venham a ser efetuadas [45]. A figura 3.6[3] representa um exemplo da operação deste algoritmo.

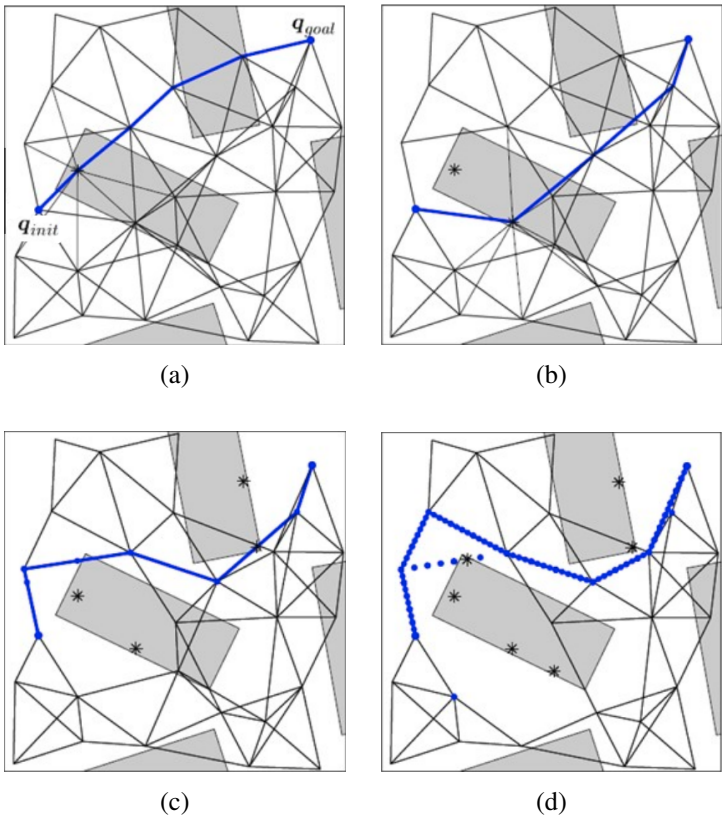


Figura 3.6: Exemplo de funcionamento do algoritmo *LazyPRM*

Probabilistic RoadMap star (*PRM**)

O algoritmo *PRM** é uma versão do algoritmo *PRM* que, a cada estado amostrado do *roadmap*, conecta os seus nós segundo uma função que garante o melhor caminho a construído, o que consequentemente garante soluções ótimas.

A cada iteração do algoritmo, os novos espaços amostrados são conectados ao *roadmap* através de uma função que avalia os restantes estados que estão contidos no interior de uma esfera de raio k. Este raio tem origem no novo nó amostrado e é uma função do número de nós na árvore, n, sendo definido pela equação 3.1, onde γ é um parâmetro baseado nas caraterísticas do ambiente e d é a dimensão do C_{space}.

$$k = \gamma \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}} \tag{3.1}$$

[3]Fonte: http://slideplayer.com/slide/6176214

Todos os nós contidos na esfera de raio k são então guardados num conjunto Q_{near}. De seguida, é efetuada uma pesquisa em Q_{near} pelo estado que representa o nó final do caminho mais curto para q_{new}, passando este a chamar-se q_{parent}. Por último, os restantes nós de Q_{near} são ligados a q_{new}, caso a sua distância a q_{parent} seja superior à sua distância a q_{new}, o que aumenta a conectividade dos nós no interior do raio k [46, 39]. As imagens da figura 3.9, alusiva ao algoritmo *RRT**, demonstram as quatro fases deste algoritmo.

Com este procedimento, à medida que o número de amostras aumenta, o tamanho de k diminui, pelo que, sendo a fase de construção do *roadmap* e a fase de pesquisa iniciadas simultaneamente, é assegurado que o algoritmo é ótimo [27].

Lazy Probabilistic RoadMap star (*LazyPRM**)

O algoritmo *LazyPRM** é a versão ótima (*star*) do algoritmo *LazyPRM*. O seu funcionamento é bastante semelhante ao da sua vertente original, sendo a grande diferença verificada no momento da amostragem do novo estado. A figura 3.7 apresenta um exemplo do funcionamento do algoritmo.

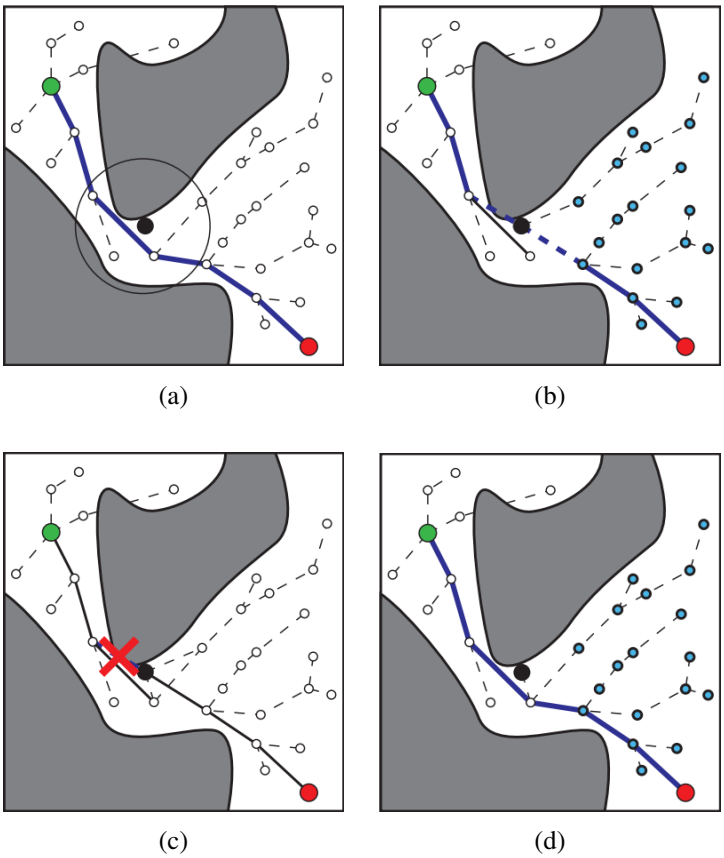


Figura 3.7: Exemplo de funcionamento do algoritmo *LazyPRM** [47]

Em (a) um novo estado é amostrado (ponto negro). Contrariamente ao algoritmo original, onde este nó seria ligado a todos aqueles no interior de um dado raio, nesta versão apenas uma ligação

é feita. Assim, todos os estados no interior da circunferência centrada no novo estado amostrado são tomados como candidatos para a ele se conectarem. Em (b) é efetuada a ligação entre o novo estado e aquele que se encontra à menor distância dele, formando assim um novo caminho entre a configuração inicial e a final. De notar que no algoritmo *LazyPRM*, todo o C_{space} é amostrado, havendo apenas a verificação de colisões na fase de pesquisa pelo melhor caminho. [47]. Em (c) é possível verificar que, nas redondezas do nó amostrado, as conexões foram novamente efetuadas, uma vez que alguns nós se encontram mais perto do novo nó do que do nó pai. No entanto, uma vez que o caminho amostrado não é viável, (d) os caminhos originais são repostos, assim como a solução inicial. O novo nó é então conectado aquele que se encontra mais próximo dele e cujo caminho é viável.

SPArse Roadmap Spanner algorithm (*SPARS*)

SPARS é um algoritmo que constrói em paralelo dois mapas, o *sparse*, *S*, e o denso, *D*, sendo este último constituído pelas configurações do C_{free} amostradas aleatoriamente. São posteriormente adicionadas a *S* as configurações de D através de um algoritmo que opera de forma similar ao *PRM* baseado em mapas de visibilidade [48]. Tem várias propriedades a ser otimizadas e é caraterizado por um critério que define o fim do algoritmo [40, 49].

SPArse Roadmap Spanner algorithm two (*SPARStwo*)

SPARStwo é uma variante do algoritmo *SPARS* que remove a dependência da existência do mapa denso *D* da sua vertente original. Funciona segundo mecânicas semelhantes mas usa uma abordagem diferente para a identificação de amostras e dos caminhos nelas construídos [40].

3.2.3 Planeadores *single-query*

Os algoritmos dos planeadores *single-query* baseiam-se na construção de árvores de estados conectadas por movimentos válidos, sendo as diferenças entre eles as abordagens e técnicas usadas para controlar como e quando a árvore se expande. Além de árvores que crescem a partir de um só estado, existem algoritmos que consideram a criação de duas árvores, onde a primeira cresce através do estado inicial e a segunda através do estado final, pelo que o seu objetivo é encontrar um nó comum a ambas as árvores, momento em que é descoberto um caminho que as une. São de seguida apresentados os algoritmos deste tipo presentes na implementação da *OMPL* do *MoveIt!*.

Rapidly-exploring Random Tree (*RRT*)

RRT é um algoritmo desenhado para procura de caminhos em espaços de elevadas dimensões. Baseia-se na construção incremental de árvores, através da ligação de espaços amostrados aleatoriamente ao vértice mais próximo da árvore, de tal forma que a distância entre os espaços amostrados e a árvore vai diminuindo sucessivamente à medida que são realizadas várias iterações

do método. Após a construção da árvore, é então pesquisado o caminho mais curto de entre todas as ramificações que ligam o ponto inicial ao ponto final [27, 39]. A figura 3.8[4] apresenta um exemplo da construção do *roadmap* neste algoritmo.

Figura 3.8: Exemplo de sucessivas iterações da construção da árvore segundo o algoritmo *RRT*

Rapidly-exploring Random Tree Connect (*RRTConnect*)

RRTConnect é um algoritmo bidirecional que aplica duas vezes o algoritmo *RRT*, partindo uma árvore do ponto inicial e outra do ponto final. O algoritmo é encerrado quando é encontrado um nó comum às 2 árvores [27, 39].

Rapidly-exploring Random Tree star (*RRT**)

*RRT** é um algoritmo semelhante ao *PRM** mas aplicado para árvores em vez de *roadmaps*. Baseia-se igualmente na equação 3.1, sendo ilustrado na figura 3.9 um exemplo da construção deste algoritmo. Em (a), após o novo nó q_{new} ser amostrado, são (b) identificados todos os nós contidos na esfera de raio k e guardados num conjunto Q_{near}. (c) É efetuada uma pesquisa em Q_{near} pelo nó que apresenta o caminho mais curto para q_{new}, passando esse nó a chamar-se q_{parent}. (d) Por último, os restantes nós de Q_{near} são ligados a q_{new} caso a sua distância a q_{parent} seja superior à sua distância a q_{new}, o que aumenta a conectividade dos nós no interior do raio k [46, 39]. Foi provado que este algoritmo é ótimo [27, 39].

De forma resumida, a vantagem deste algoritmo relativamente à sua vertente original é o facto de a árvore de estados ser atualizada cada vez que novos nós lhe são adicionados. De notar também que as árvores formadas pelo algoritmo *RRT** são mais alongadas, uma vez que ramos maiores evidenciam caminhos mais retos e fáceis de percorrer. A figura 3.10[5] demonstra um exemplo das árvores de estados criadas pelos dois algoritmos.

Lower Bound Tree Rapidly-exploring Random Tree (*LBTRRT*)

O algoritmo *LBTRRT* consiste na manutenção de duas árvores que, apesar de partilharem os mesmo vértices, apresentam diferentes conexões entre eles. Ambas apresentam caraterísticas

[4] Fonte: https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

[5] Fonte: https://www.cc.gatech.edu/ dellaert/11S-AI/Topics/Entries/2011/2/21_PRM,_RRT,_and_RRT__files/06-RRT.pdf

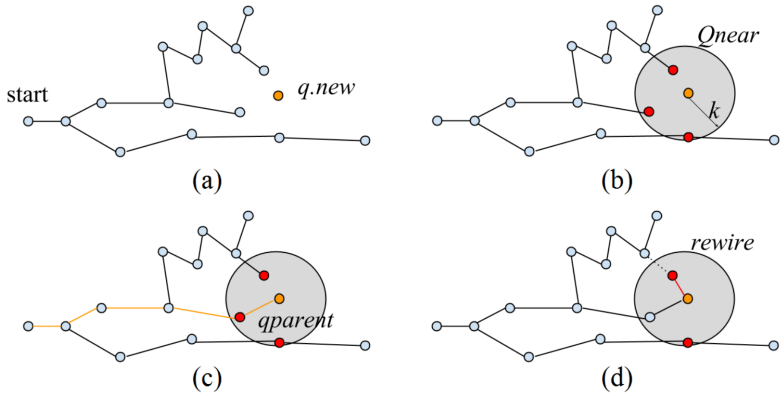


Figura 3.9: Exemplo do algoritmo *RRT** [39]

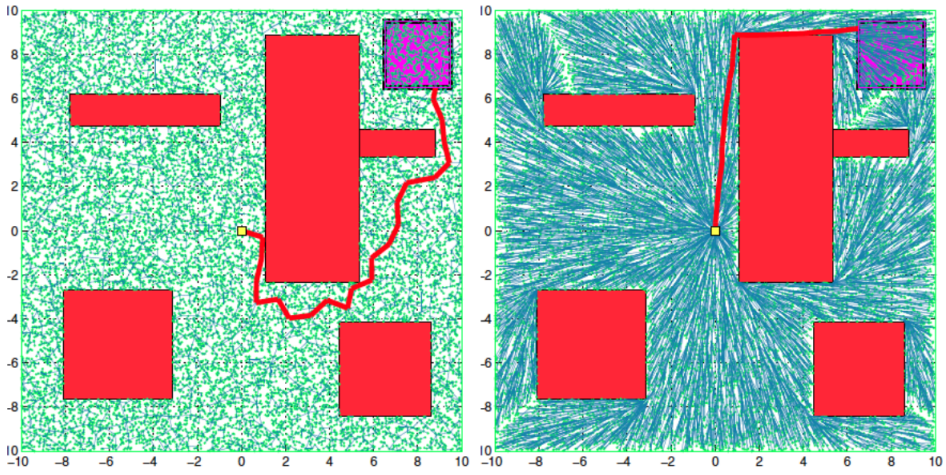


Figura 3.10: Comparação entre as árvores criadas pelos algoritmos *RRT* (esquerda) e *RRT** (direita)

específicas: a primeira árvore é caraterizada pelo facto de nem todos os seus caminhos serem viáveis, apesar de que estes apresentam um menor valor da função de custo na procura do caminho; a segunda árvore apresenta caminhos sempre viáveis, apesar de que estes têm um valor da função de custo pelo menos (1+ε) superior ao valor mais baixo (*lower bound*) da função de custo dos caminhos da primeira árvore.

Transition-based Rapidly-exploring Random Tree (*TRRT*)

O algoritmo *TRRT* é uma extensão do original *RRT* na medida em que adiciona um teste de transição estocástica para redirecionar a exploração e crescimento da árvore no sentido das suas regiões de mais baixo custo [50]. Este teste de transição é baseado no critério de Metropolis, tipicamente usado nos métodos de otimização de Monte Carlo [51]. Tem como objetivo aceitar ou rejeitar estados candidatos a ligarem-se ao novo nó da árvore, baseando-se na variação de custo associado ao movimento local.

Bi-directional Transition-based Rapidly-exploring Random Tree (*BiTRRT*)

O algoritmo *BiTRRT* é a versão bidirecional do seu original *TRRT*, onde duas *TRRTs* crescem, uma a partir da configuração inicial e outra a partir da configuração final, tendo como objetivo encontrarem-se.

Expansive Space Trees (*EST*)

EST é um algoritmo que tem como objetivo detetar as áreas menos exploradas de um *roadmap*. Consiste numa primeira fase de expansão, e na fase de conexão. Na primeira fase são amostrados estados aleatórios e retidos aqueles que podem ser conectados a uma configuração inicial q_{init} ou a uma configuração final q_{goal}. Desta forma, o algoritmo evolui de forma a apenas explorar as zonas de interesse para alcançar a solução, evitando a computação do *roadmap* para o restante C_{space}. Na segunda fase é construída uma árvore que liga a configuração inicial à final. *p* [27, 52].

Bi-directional Expansive Space Trees (*BiEST*)

Bi-directional Expansive Space Trees é a versão bidirecional do algoritmo *EST*, ou seja, na fase de conexão, são construídas duas árvores através das configurações inicial e final. A solução é encontrada quando as árvores se intercetarem.

Projection Expansive Space Trees (*ProjEST*)

ProjEST é um algoritmo baseado na sua versão original, mas que deteta as áreas menos exploradas do espaço através do uso de uma grelha imposta na projeção do espaço de estados. Caso não haja nenhuma projeção definida, o algoritmo usa a projeção predefinida associada ao espaço de estados [40].

Single-query Bi-directional Lazy collision checking planner (*SBL*)

SBL é um algoritmo baseado em árvores, com a mesma estratégia de expansão do *EST*, e também munido das propriedades *lazy* e bidirecional, já anteriormente abordadas. De forma a registar a exploração do C_{space}, o algoritmo usa uma grelha como estrutura de dados que contém os estados que já foram previamente visitados. Esta grelha é também usada aquando da expansão da árvore, sendo que células que foram menos vezes visitadas têm uma maior probabilidade de serem selecionadas. Esta grelha é estabelecida numa projeção do espaço de estados [40].

Kinodynamic motion Planning Interior-Exterior Cell Exploration (*KPIECE*)

KPIECE é um algoritmo desenvolvido para sistemas com dinâmicas complexas que toma partido do modelo de espaço de estados do sistema para construir uma árvore de movimentos. Tem como entradas o estado inicial e as variáveis de controlo e como saída o próximo estado do sistema. De forma resumida, o algoritmo consiste em verificar quais os estados que representam

movimentos exteriores à sua árvore e que podem ser alcançados segundo o movimento do robô num dado instante de tempo. Em cada iteração a árvore é expandida, segundo restrições dinâmicas que limitam a sua gama de estados atingíveis, até que se atinja o estado final pretendido [27, 53].

Bi-directional Kinodynamic motion Planning Interior-Exterior Cell Exploration (*BKPIECE*)

O algoritmo *BKPIECE* é a versão bidirecional do algoritmo *KPIECE*. Baseia-se no mesmo conceito de expansão da árvore, sendo que a única diferença é o facto de serem construídas duas árvores: uma que nasce na configuração inicial e outra na configuração final. A solução é encontrada quando as árvores se intercetarem.

Lazy Bi-directional Kinodynamic motion Planning Interior-Exterior Cell Exploration (*LBKPIECE*)

O algoritmo *LBKPIECE* é a versão *lazy* do algoritmo *BKPIECE*. Baseia-se nos conceitos já abordados anteriormente para outros algoritmos.

Search Tree with Resolution Independent Density Estimation (*STRIDE*)

STRIDE é um algoritmo que tem como objetivo detetar as áreas menos exploradas do mapa através do uso de uma estrutura *nearest-neighbor* do tipo *Geometric Near-neighbour Access Tree* (*GNAT*) [54]. De forma resumida, uma *GNAT* é uma árvore de nós em que cada nó pode representar uma ou mais configurações do robô. Além destes nós, contém uma função de distância que permite comparar as diferentes configurações [55].

Path-Directed Subdivision Trees (*PDST*)

PDST é um algoritmo que deteta as áreas menos exploradas do espaço de configuração através do uso de uma partição binária de uma projeção do espaço de estados. A exploração neste algoritmo é direcionada para células de maiores dimensões que ainda não tenham a elas conectados muitos caminhos. Contrariamente à maioria dos algoritmos baseados em árvores que se expandem a partir de uma extremidade aleatória de um caminho, o *PDST* expande-se a partir de uma ponto aleatório ao longo de um segmento selecionado deterministicamente [40].

Fast Marching Tree (*FMT*)

FMT é um algoritmo desenhado para diminuir o número de verificações de colisões e é particularmente eficiente em configurações com várias dimensões e munidas de obstáculos. Constrói árvores em forma de disco e também tem como base a propriedade *lazy*, isto é, apenas efetua a verificação de colisões após a construção da árvore. É inspirado no *Fast Marching Method*, um algoritmo que se baseia na premissa de que novos pontos amostrados devem ser adicionados à árvore de tal forma que os caminhos construídos cresçam no sentido de expandir sempre a árvore [56]. A figura 3.11 apresenta um exemplo da expansão de uma árvore criada por este algoritmo.

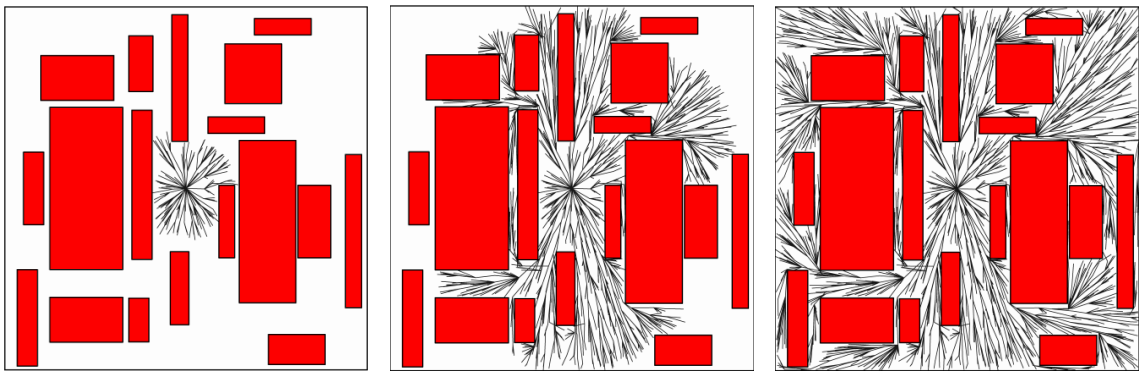


Figura 3.11: Exemplo de expansão de uma árvore do algoritmo *FMT** [56]

***Bi-directional Fast Marching Tree* (*BFMT*)**

BFMT é a versão bidirecional do algoritmo *FMT**, onde é construída uma árvore através da configuração inicial e outra através da configuração final [57].

A implementação de todos estes algoritmos no *MoveIt!* é transparente para o utilizador, o que constitui uma grande vantagem, dado que este apenas tem que selecionar qual pretende utilizar. Ainda assim, o código fonte da sua implementação encontra-se disponível na documentação disponibilizada. Existem, no entanto, parâmetros configuráveis para os planeadores, sendo alguns comuns a todos e outros específicos de algumas implementações. O estudo das combinações e configurações de planeadores é efetuado no capítulo 5.

O seguinte capítulo aborda a implementação da plataforma AGROB V16 no *Gazebo*, assim como a integração entre o braço robótico e o *MoveIt!*.

Capítulo 4

Simulação da plataforma AGROB V16

Neste capítulo são abordados os procedimentos inerentes à implementação do cenário de simulação que serviu de ponto de partida para o *benchmarking* dos algoritmos de planeamento de trajetórias. Inicialmente é descrita a implementação da plataforma AGROB V16 no ambiente de simulação *Gazebo*, sendo de seguida descrita a integração entre o braço robótico e o *software MoveIt!*.

4.1 AGROB V16

A plataforma robótica AGROB V16 foi desenvolvida com o intuito de ser escalável, comportando a possibilidade de endereçar sensores e atuadores mediante a necessidade de novas funcionalidades. É constituída por três principais elementos: o robô *Husky*, o braço robótico *Robotis Manipulator-H* e a torre, onde se situam sensores, unidades de processamento e interfaces. A integração destes três elementos e implementação no simulador *Gazebo* é de seguida apresentada.

4.1.1 *Husky*

O *Husky* (figura 4.1[1]) é uma plataforma robótica semelhante a um pequeno carro, cujo *software* foi desenvolvido inteiramente segundo as convenções do *ROS*. É bastante comum em projetos de investigação, uma vez que é um robô pioneiro do *ROS* e para o qual existe muito *software* disponível. Uma das suas aplicações mais recentes é precisamente na área da poda, no projeto de poda das vinhas francesas [13] referido no subcapítulo 2.1.1.

Comporta como dimensões 99cm de comprimento, 67cm de largura e 39cm de altura, pesando 50kgs e com um *payload* máximo de 75kgs. Têm uma estrutura metálica com rodas semelhantes às de um trator, o que lhe confere a robustez necessária para operar em ambientes agrestes e pedregosos, como é o caso do ambiente de vinha de encosta.

Atualmente encontra-se desenvolvido o *metapackage husky*[2] (disponível para a versão *Kinetic* do *ROS*, entre outras), que contém vários *packages ROS* com as bibliotecas necessárias para

[1] https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/
[2] https://github.com/husky/husky

Figura 4.1: Robô *Husky*, base da plataforma AGROB V16

simular e operar o *Husky* em ambiente real e nos ambientes de simulação *Gazebo* e *RViz*. Nesta dissertação, o *package husky_gazebo* (contido no *metapackage husky*) toma um papel fundamental, permitindo simular este robô no *Gazebo*, como se pode observar na figura 4.2. De notar que o eixo vermelho corresponde aos eixo dos *xx*, o verde ao dos *yy* e o azul ao dos *zz*. Esta convenção mantém-se para toda e qualquer imagem referente ao *Gazebo* que daqui para a frente seja apresentada.

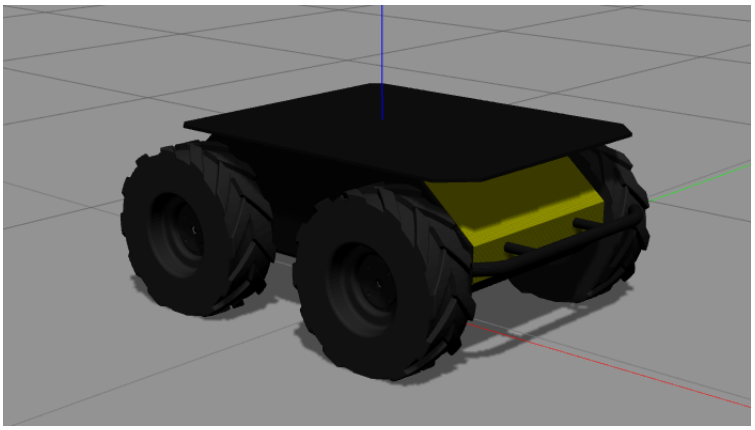
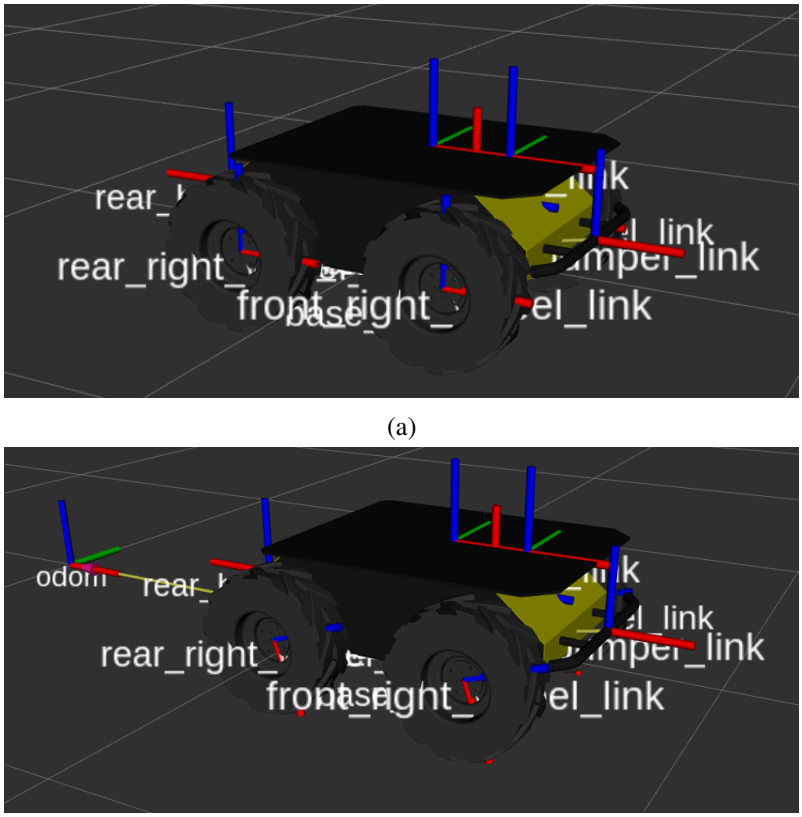


Figura 4.2: Robô *Husky* simulado em *Gazebo*

No *ROS*, os nomes dos componentes de cada robô partilham também o seu nome com referenciais (*frames*), que representam a sua origem e orientação no espaço. Uma forma de visualizar os referenciais do robô e suas interações é através do *package rqt* (não são apresentadas imagens dos referenciais devido às suas dimensões excederem as de uma página A4), que permite ainda visualizar nós e tópicos (figura 3.1), entre outra funcionalidades.

Outra solução é o ambiente de simulação *RViz*, no qual podem ser visualizadas, em tempo real, as posições exatas destes referenciais, assim como as respetivas orientações. Alguns referenciais são fixos, isto é, mantém sempre a mesma orientação e a sua posição é apenas transladada caso o robô altere a sua posição e o seu referencial pai se mova devido a este movimento. No entanto, o mesmo não acontece com alguns referenciais que representam componentes com mais graus de

liberdade, como é o caso das rodas do *Husky*. Na figura figura 4.3 apresenta-se um exemplo de duas posições do robô, onde é possível observar a evolução temporal dos referenciais.



(a)

(b)

Figura 4.3: Exemplo das posições dos referenciais do *Husky* a) na sua posição inicial e b) após um deslocamento para a frente de alguns metros

Estes eixos também podem ser visualizados no *Gazebo*, ainda que não sejam tão percetíveis como neste simulador.

Na plataforma AGROB V16 não consta a placa metálica que serve como base do topo do *Husky* e que pode ser observada nas figuras 4.1 e 4.2. Para que a simulação representasse corretamente o robô real, foram alterados os ficheiros *URDF* do *package husky_description*, onde estão descritas todas as componentes do robô e respetivas ligações. Esta placa metálica corresponde ao componente *top_plate_link*, pelo que a sua remoção consiste apenas na eliminação da ligação entre este componente e o seu pai, *base_link*. O resultado final da remoção da placa é o robô da figura 4.4, com o seu interior oco, onde, no robô real, se encontram as suas baterias.

De seguida é apresentado o manipulador e os restantes componentes que auxiliam a sua montagem no *Husky*.

4.1.2 *Robotis Manipulator-H*

O *Robotis Manipulator-H* foi previamente apresentado no subcapítulo 2.2.2, pelo que é sabido que o seu peso é suportado pelo *Husky*. Existe um *metapackage ROS* , desenvolvido pelo seu

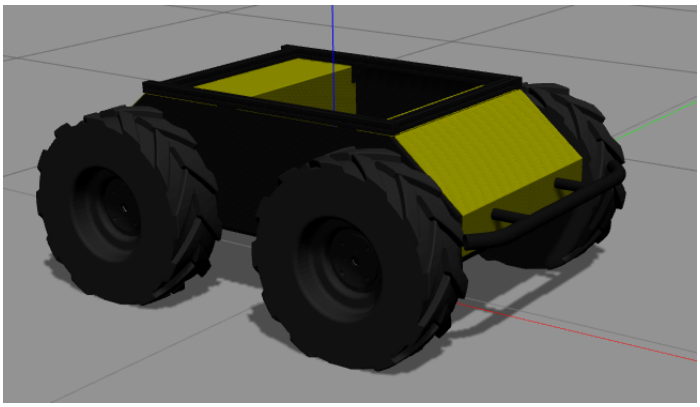


Figura 4.4: Robô *Husky* simulado em *Gazebo* sem a placa metálica no seu topo

fornecedor, a *Robotis*, para que o robô possa ser programado e operado em ambiente real ou nos simuladores já indicados, o *ROBOTIS-MANIPULATOR-H*[3]. Entre os vários *packages* de que dispõe, destacam-se o *manipulator_h_gazebo* e o *manipulator_h_description*. O primeiro permite simular o manipulador no *Gazebo* através da descrição contida nos ficheiros *URDF* do segundo. A figura 4.5 mostra a representação do manipulador no *Gazebo*.

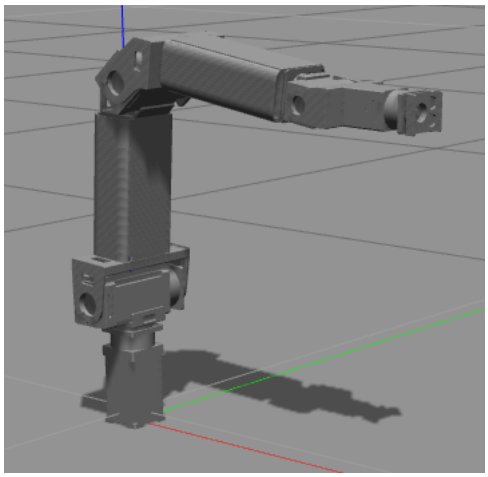


Figura 4.5: Braço robótico *Robotis Manipulator-H* no simulador *Gazebo*

A sua descrição no ficheiro *URDF* permite descrever este manipulador como uma série de componentes ligados entre si através de juntas rotativas. Existem no total sete componentes, desde o *link1* até ao *link6* e *endlink*, a pequena peça que se encontra na extremidade do manipulador. As seis ligações entre os componentes conferem os seis graus de liberdade de que é dotado este robô.

No AGROB V16 o manipulador não se encontra diretamente preso ao topo do *Husky*. Na figura 1.3 é possível observar que o braço se encontra assente no topo de um disco de metal, que por sua vez se encontra preso a um suporte de metal. Este foi construído para conferir mais altura ao volume de trabalho do manipulador, assim como para conter toda a cablagem a ele associada e permitir ainda que se ligue o computador associado à torre, ou apenas o manipulador. Esta última

[3]https://github.com/ROBOTIS-GIT/ROBOTIS-MANIPULATOR-H/tree/kinetic-devel

função é possível através dos botões instalados num dos lados deste suporte. A figura 4.6, mais recente que a 1.3, apresenta já uma versão deste suporte com duas faces isoladas, estando os botões dispostos numa dessas faces.

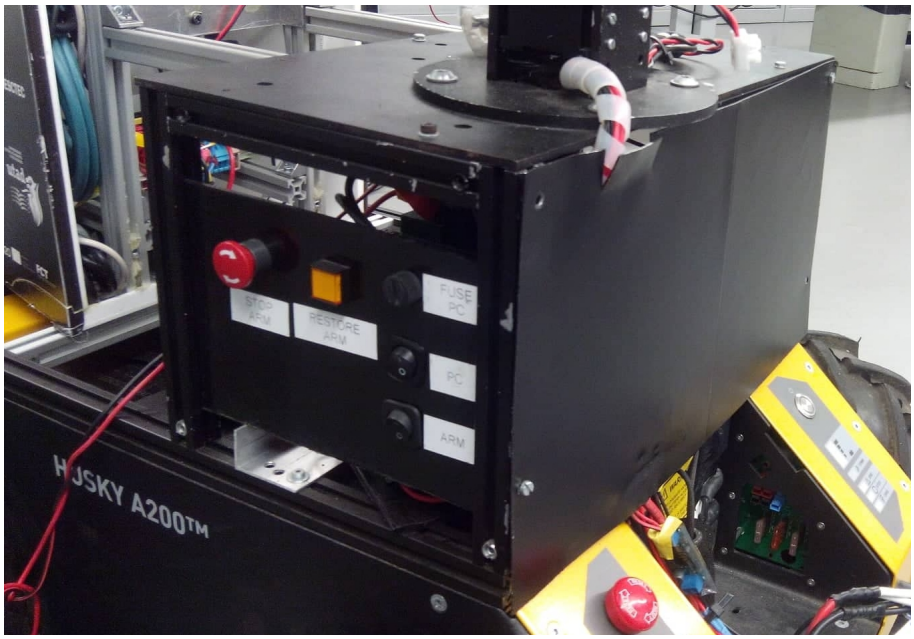


Figura 4.6: Faces laterais do suporte do manipulador

Como forma de representar todo este suporte em *Gazebo*, foram construídos blocos em forma de paralelepípedo e de cilindro, com formatos idênticos aos do robô real. Para isso, foram efetuadas medições com recurso a uma fita métrica, que permitiram afirmar que o suporte tem 22,5*cm* de comprimento, 42*cm* de largura e 20,2*cm* de altura. Por sua vez, o cilindro tem 7,5*mm* de altura e 8*cm* de raio. Para tornar nítida a distinção entre este dois componentes na simulação, ao paralelepípedo foi conferida a cor laranja e ao cilindro a azul.

Por fim, como forma de fixar o manipulador e respetivas bases ao *Husky*, foram definidas juntas fixas entre os referenciais *base_link* (definido no centro do *Husky*) e os referenciais dos restantes componentes: referencial *box* para o paralelepípedo, *cylinder* para o cilindro e *link1*, que representa o primeiro componente do manipulador. A calibração das distâncias entre referenciais permitiu chegar à versão do AGROB V16 representada na figura 4.7.

Após a análise efetuada no subcapítulo 2.2.3, e tal como já referido nesse mesmo capítulo, a escolha para o *end effector* do manipulador recaiu sobre uma tesoura de poda semelhante à da figura 2.5. Uma tesoura deste tipo já tinha sido previamente projetada para um projeto do INESC TEC que tinha como objetivo desenhar um *drone* que conseguisse podar árvores durante o seu voo. Ainda que esse projeto não tivesse sido implementado, o desenho da tesoura foi efetuado e completo no *software SolidWorks*, e, uma vez que as suas dimensões são compatíveis com as do manipulador, este modelo foi o ponto de partida para a implementação da tesoura neste projeto.

De forma resumida, o *SolidWorks* permite desenvolver todo o tipo de peças mecânicas e simulá-las de modo a prever o seu comportamento em ambiente real. Ainda que este programa

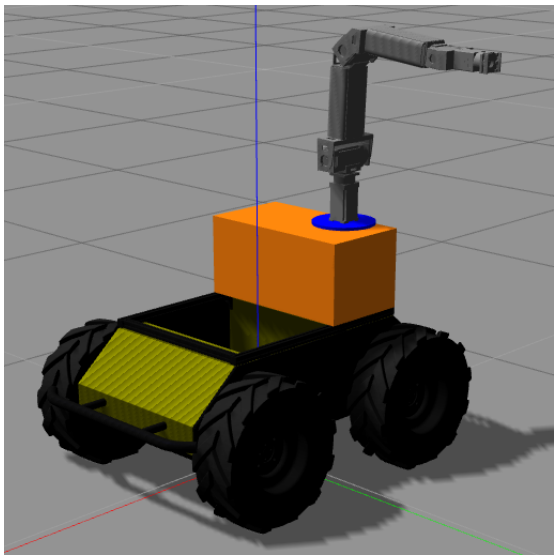


Figura 4.7: Plataforma AGROB V16 agora constituída pelo robô *Husky* e pelo manipulador *Robotis Manipulator-H*

apenas esteja disponível para o sistema operativo *Windows*, e que os seus projetos e peças não possam ser exportados em nenhum tipo de ficheiro compatível com o *ROS*, foi encontrada uma solução para empregar a peça desenvolvida no simulador *Gazebo*, o programa *SolidWorks to URDF Exporter*.

O *SolidWorks to URDF Exporter*[4] é um *plugin* compatível com *SolidWorks*, para versões superiores à lançada em 2012. Permite exportar peças ou montagens efetuadas neste programa para *packages* que contém a descrição destes componentes em formato *URDF*, compatível com *ROS* e seus ambientes de simulação. É um *software open-source* desenvolvido por Stephen Brawner em 2012, que tem como objetivo ultrapassar a dificuldade que é criar peças compatíveis com os formatos suportados pelo *ROS*. Ainda que não tenha sido melhorado após o seu ano de lançamento, o *software* permitir exportar peças individualmente, sendo que o mesmo não se pode afirmar no que toca a exportar modelos complexos (montagens) que consistem em várias peças, desde pequenos parafusos até grande estruturas metálicas.

O modelo da tesoura de poda consiste numa montagem complexa devido a todas as pequenas componentes que o constituem. Cada um destas componentes apresenta-se representada com diferentes cores na figura 4.8, retirada do *SolidWorks*.

De notar que o eixo *Origin_global* representa a posição em que se pretende posicionar o *end effector* no ponto de corte. A distância vertical (no eixo dos *zz*) que atravessa este eixo é de 1,5*cm* e representa assim o valor máximo que o diâmetro das vides pode comportar de modo a poder ser cortado por esta tesoura.

Nesta representação da tesoura, as peças estão conectadas entre si e apenas são amovíveis aquelas que são dotadas desse tipo de dinâmica, como é o caso de uma das lâminas da tesoura (a que não é fixa). No entanto, a construção de uma tesoura deste tipo sugere que as suas lâminas

[4] http://wiki.ros.org/sw_urdf_exporter

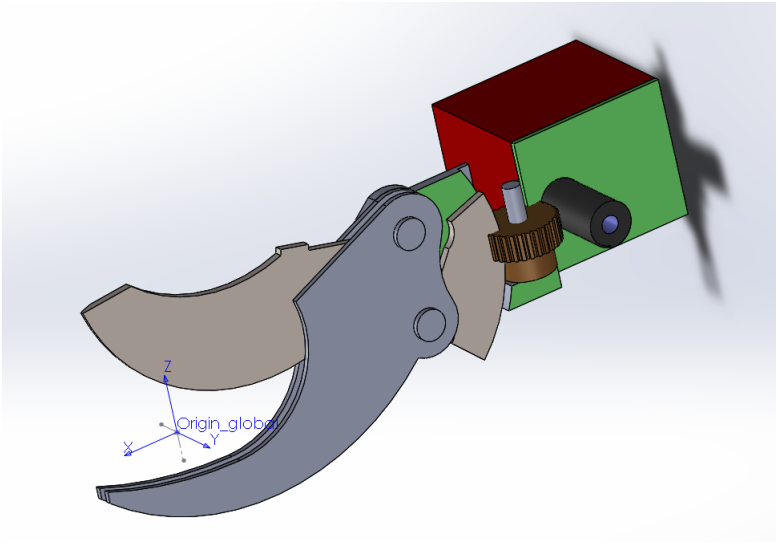


Figura 4.8: Modelo da tesoura de poda desenhado no *software SolidWorks*

se encontrem numa posição naturalmente aberta, e que apenas sejam fechadas quando se efetue um corte. Uma vez que as operações de corte não estão otimizadas no *Gazebo*, isto é, como uma operação de corte acabaria por resultar numa colisão entre as lâminas e o tronco, com resultados que não representariam corretamente a realidade, optou-se por não dotar as lâminas de qualquer movimento. Uma forma de o fazer é através da conversão do modelo de montagem da tesoura num modelo que represente a tesoura como um todo, isto é, como um só componente cujas partes não são dotadas de qualquer tipo de dinâmica.

Tal como já referido anteriormente, o *plugin* permite exportar peças ou montagens, sendo que o ficheiro original da tesoura era uma montagem e não um peça. Este tipo de ficheiros, em formato *SolidWorks*, designam-se de *SolidWorks Assembly* (*sldasm*), ao passo que os ficheiros das peças são gravados no formato *SolidWorks Part* (*sldprt*). Assim, o ficheiro original *sldasm* da tesoura foi convertido no formato *sldprt* e a tesoura foi exportada para o ambiente *ROS* através deste *plugin*. Finalmente, como forma de assemblar esta peça ao restante manipulador, foi definida uma junta fixa entre este componente, ao qual foi dado o nome de *cutting_tool* na simulação, e o componente *end_link*. O resultado final da montagem desta assemblagem encontra-se na figura 4.9.

Apesar desta representação da tesoura ser possível de posicionar em qualquer posição válida, há pormenores que devem ser destacados relativamente aos modelos exportados através deste *plugin*. A ferramenta de poda apresenta a particularidade de ser completamente branca, o que não representa exatamente o seu modelo em *SolidWorks*. Isto deve-se a uma das limitações do *plugin*, que não consegue exportar a cor dos objetos. A objetos sem cor, o *Gazebo* atribui automaticamente a cor branca, daí que a sua representação neste simulador tenha tomado essa cor. Ainda que a cor possa ser alterada manualmente no ficheiro *URDF*, a cor foi branca foi mantida pois contrasta com a cor escura do modelo da vinha utilizado para efetuar os testes.

Além da calibração que tem que de ser efetuada para posicionar a ferramenta na posição correta, foi necessário definir o referencial da peça de modo a que qualquer posicionamento da

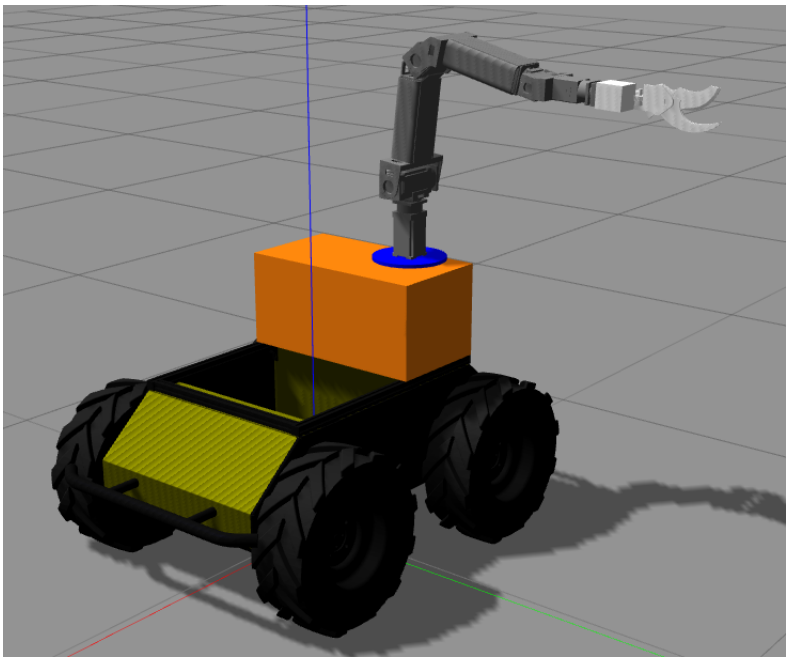


Figura 4.9: Plataforma AGROB V16 agora constituída pelo robô *Husky* e pelo manipulador *Robotis Manipulator-H* com a tesoura de poda como *end effector*

mesma se reflita em levar a origem deste referencial para um ponto definido. Relativamente à posição dos eixos de coordenadas, é possível observar na figura 4.10 que os eixos do manipulador foram definidos de tal forma que o eixo dos *xx* aponta na direção que coincide com aquela para a qual aponta o manipulador.

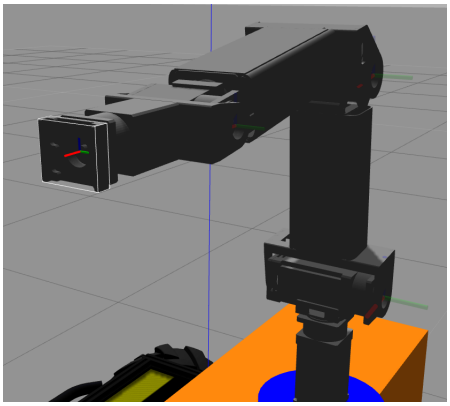


Figura 4.10: Eixos de coordenadas no componente *end_link* do manipulador

Assim, como forma de manter esta convenção, os eixos de coordenadas da tesoura de poda foram alterados no *plugin* de forma a manter a mesma direção para o qual apontam os eixos do componente ao qual ela está acoplada. Relativamente à posição do seu eixo, esta deve refletir a zona onde se deve posicionar a tesoura, com as lâminas abertas, antes de efetuar o corte. Assim, esta posição foi também calibrada, de modo a melhor representar a zona onde é normalmente efetuado um corte por uma pessoa. A figura 4.11 demonstra em detalhe a posição do referencial

na tesoura.

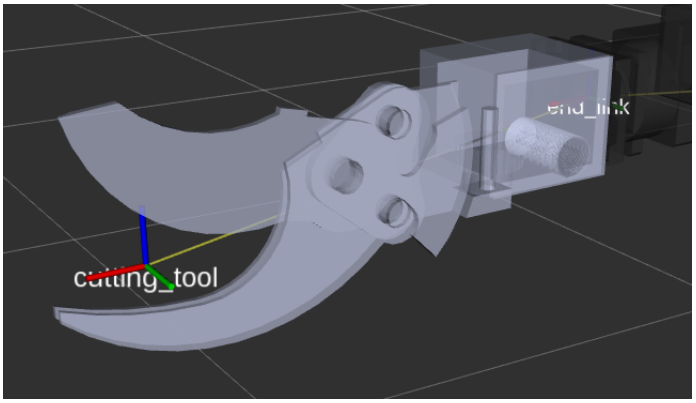


Figura 4.11: Eixos de coordenadas a tesoura de poda quando assemblada no manipulador

Em relação ao facto de as lâminas não poderem ser operadas na simulação, é de esperar que em ambiente real o corte seja efetuado logo após o posicionamento da tesoura no local de corte.

De seguida é apresentada a torre e os seus elementos que se revelaram importantes no âmbito desta dissertação.

4.1.3 Torre

Na torre de controlo estão contidos todos os elementos de sensorização e de rede, que permitem a interação do AGROB V16 com o meio envolvente. Encontra-se montada na parte frontal do *Husky* e está dividida em duas peças: um prisma quadrangular e um paralelepípedo com cantos redondos, preso no topo do prisma. O conjunto mede cerca de 95cm de altura e 45cm de largura, contendo vários sensores acoplados ao longo de toda a estrutura, desde a parte inferior do prisma, até àqueles que estão contidos no interior do paralelepípedo. Contém ainda um computador, operável através de um ecrã táctil, que controla todo o fluxo de dados.

No âmbito desta dissertação, destaca-se o *laser Hokuyo UST-10LX*[5], um dispositivo preciso e de alta velocidade que está otimizado para a deteção de obstáculos e localização espacial em sistemas robóticos autónomos. Possui um ângulo de deteção de 270º, com uma resolução de 0,25º e uma distância (regulável) máxima de deteção de 30m, permitindo atingir precisões milimétricas nas suas medições, imprescindíveis na deteção de obstáculos de pequenas dimensões, como é o caso das vides. A figura 4.12 apresenta uma imagem deste modelo. De notar que o *laser* efetua medições no plano perpendicular à direção para onde aponta, isto é, no plano da sua base laranja.

Este dispositivo encontra-se montado na parte superior do prisma, como é possível observar na figura 4.13.

Além de auxiliar o robô a localizar-se no meio, o facto deste *laser* detetar qualquer obstáculo nas imediações do robô, à medida que este se desloca, permite que todo o ambiente seja mapeado numa estrutura tridimensional, fulcral para o planeamento de trajetórias do manipulador. Este processo é descrito em maior detalhe no subcapítulo 4.2.2.

[5]http://www.hokuyo-usa.com/products/scanning-laser-rangefinders/ust-10lx

Figura 4.12: *Laser Hokuyo UST-10LX*

A implementação deste sensor na simulação consiste em criar um novo componente que represente o sensor, acoplá-lo à restante plataforma e disponibilizar um tópico onde são publicados os valores das suas medidas.

Uma vez que este sensor já foi utilizado em projetos anteriores do INESC TEC para a localização do AGROB V16 no ambiente, a posição e orientação exatas do sensor na simulação já foram previamente implementadas. Nesses projetos, a posição e orientação do *laser* foram definidas relativamente ao referencial de um outro *laser* que se encontra acoplado à parte inferior da torre, que por sua vez se encontra definido em relação ao referencial *base_link* do *Husky*. Uma vez que esse *laser* não é usado de nenhuma forma para auxiliar a deteção de obstáculos, o referencial do *laser Hokuyo* foi definido diretamente em função do referencial *base_link*. Para isso, foram somados os valores da posição e orientação que correspondem à translação e rotação entre os referenciais *base_link* e do *laser* da base da torre aos valores da mesma operação entre os referenciais do *laser* da base da torre e do *laser Hokuyo*, resultando assim num referencial que espelha a transformação direta entre o *laser Hokuyo* e o referencial *base_link*, tal como demonstrado na figura 4.14. Ao referencial do *laser Hokuyo* foi dado o nome de *hokuyo_link*.

Ainda que a imagem desta figura tenha sido retirado do simulador *RViz*, o aspeto deste *laser* é idêntico no *Gazebo*.

O tópico onde são publicadas as mensagens com as medidas do sensor tem o nome de *vertical_laser*, sendo que as mensagens são do tipo *sensor_msgs/LaserScan*, que permitem a deteção da posição e distância do ponto encontrado, relativamente ao sensor.

Relativamente à simulação da torre, a sua implementação é a que menos representa a realidade, uma vez que não é necessário um detalhe excessivo dos seus elementos. Considere-se o caso da base manipulador. Apesar das suas dimensões terem que ser o mais reais possível, não é necessário pormenorizar o seu interior, uma vez que não é uma zona de interesse. Por outro lado,

Figura 4.13: *Laser Hokuyo UST-10LX* montado na torre da plataforma AGROB V16

uma superfície cujo interior é inalcançável pelo manipulador também garante que, aquando do planeamento de trajetórias, essas zonas nunca serão tidas em conta e, consequentemente, nunca serão atingidas. Isto comporta uma vantagem para a segurança de toda a plataforma pois garante que, para o *MoveIt!*, algumas zonas da sua estrutura da plataforma são de facto componentes sólidos cujo manipulador não pode atingir (como será explicado no próximo subcapítulo), quando na verdade são espaços perfeitamente acessíveis. Tendo isto em conta, a estrutura da torre simulada, que também é aquela que o *MoveIt!* interpreta como sendo a real, não comporta as verdadeiras dimensões do prisma retangular e de todos os seus sensores, mas sim a de um paralelepípedo que envolve todo o prisma. O único sensor que foi deixado de fora do paralelepípedo foi precisamente o *laser Hokuyo*, uma vez que a área de atuação do seu *laser* não pode estar obstruída por qualquer objeto. A figura 4.15 representa então a versão final do AGROB V16 simulado no *Gazebo*. De notar que os raios azuis representam a área de deteção do *laser*.

Uma vez apresentado o robô simulado, é de seguida apresentada a integração e interação entre o *ROS* e o *MoveIt!*.

4.2 *MoveIt!*

Tal como já foi apresentado no subcapítulo 3.1.3, o *software MoveIt!* é um *plugin* suportado pelo *ROS* que permite efetuar o planeamento de trajetórias, em espaços de configuração bidimensionais e tridimensionais, tendo em conta obstáculos neles existentes. Funciona segundo uma arquitetura bastante organizada, esquematizada segundo o diagrama da figura 4.16[6].

[6]http://moveit.ros.org/documentation/concepts/

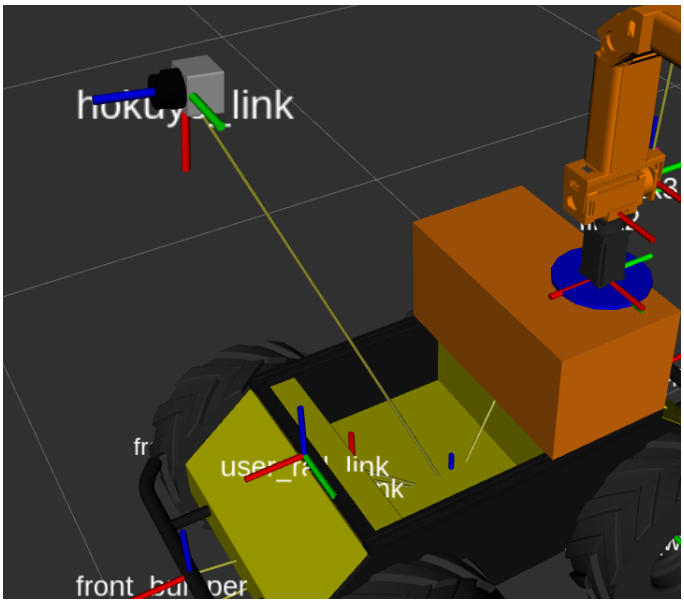


Figura 4.14: Relação entre os referenciais *hokuyo_link* e *base_link* (no interior do *Husky*)

No topo do diagrama encontra-se o bloco *ROS Param Server*, que remete para o servidor de parâmetros *ROS*, onde é carregada toda a descrição do AGROB V16 segundo os vários ficheiros *URDF* que o descrevem. A informação do robô é então enviada para o bloco central principal, o bloco *move_group* que representa o nó *ROS* lançado sempre que é corrido o *MoveIt!*. Este é responsável por interagir com o robô, subscrevendo trajetórias e publicando mensagens que permitem efetuar o *tracking* das mesmas, assim como verificar as posições das juntas do robô.

Os blocos a cinzento e as suas ligações com o bloco central representam as diferentes interações que o utilizador pode ter com o *MoveIt!*. Os blocos *move_group_interface (C++)* e *moveit_commander (Python)* são exemplos de nós que podem ser lançados e que requisitam ao manipulador ações (setas vermelhas) como a movimentação do robô (*MoveGroupAction*), o fecho da garra (*PickAction*) ou ainda a abertura da garra (*PlaceAction*). Estas mesmas ações podem ser efetuadas através do *plugin* do *MoveIt!* para o *RViz*, que funciona como um *GUI* orientado para uma mais fácil operação do robô. Esta particular janela do *RViz* é lançada conjuntamente com o nó do *MoveIt!*, e toma o aspeto da imagem da figura 4.17.

As setas a azul representam algumas das funções chamadas quando é requisitado ao manipulador que execute uma dada trajetória. As setas a verde demonstram ainda outra funcionalidade do *MoveIt!*, que é o facto de permitir que alguns objetos sejam anexados ao manipulador (*AttachedObject*), assim como declarar objetos como sendo objetos de colisão (*CollisionObject*), que entram em conta aquando do planeamento de trajetórias.

Os blocos a azul claro representam ações relacionadas com os atuadores e sensores, isto é, os comandos enviados e os dados recebidos. Cada trajetória é enviada segundo uma *JointTrajectoryAction* para cada junta do robô. Na sua forma mais simples, uma trajetória de um manipulador pode ser simplesmente entendida como sendo um conjunto de *waypoints*, onde cada *waypoint* corresponde a um conjunto de posições das juntas, que todos os controladores do braço devem

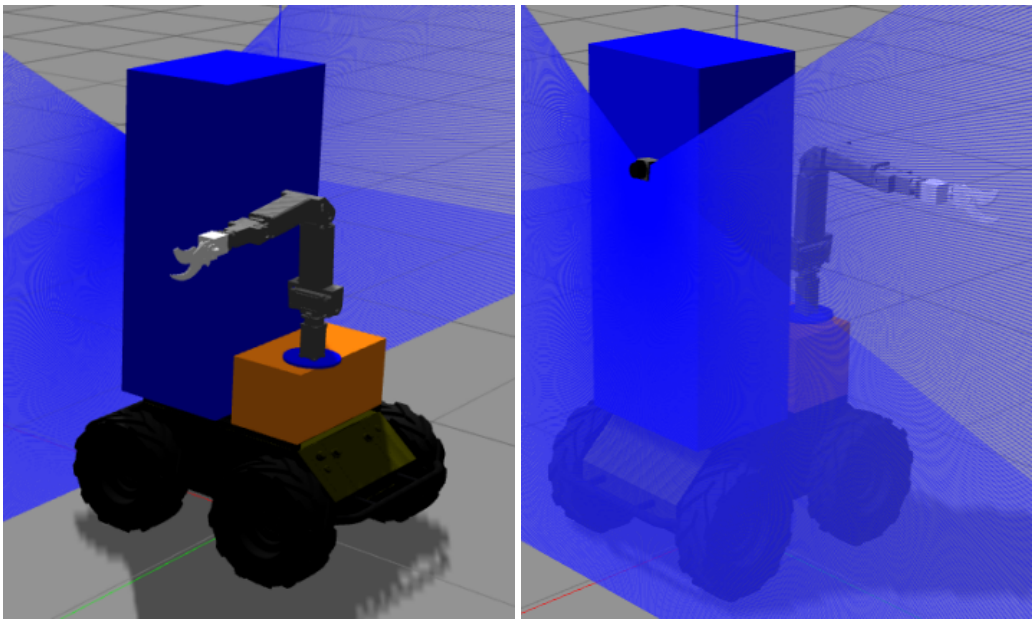


Figura 4.15: Versão final do AGROB V16 simulado em *Gazebo*

atingir antes de alterarem a sua posição para a do próximo *waypoint*. Esta ação faz o *tracking* da trajetória, garantindo, em conjunto com o nó *move_group*, que esta é seguida.

Os sensores do robô funcionam como o meio que este tem para se percecionar acerca do seu meio envolvente. Nesta dissertação têm um papel fundamental, uma vez que o robô têm que descobrir o seu meio envolvente à medida que se desloca, dado que não existe um mapa predefinido das vinhas.

Nos próximos subcapítulos são abordados os principais conceitos acerca do *MoveIt!*. Inicialmente será apresentado o *MoveIt! Setup Assistant*, seguindo-se a interação entre este *software* e os sensores de perceção espacial. Por último, são apresentadas as suas principais funcionalidades. Os detalhes da integração entre o *MoveIt!* e o *Gazebo* não são abordados uma vez que não são o foco desta dissertação.

4.2.1 *MoveIt! Setup Assistant*

Robôs complexos, como é o caso do AGROB V16, integram várias componentes, sendo que nem todas necessitam do tipo controlo que o *MoveIt!* pode proporcionar. Tal como dito acima, o *MoveIt!* tem que ter noção de todo o robô através dos seus ficheiros *URDF*. Ainda assim, é necessário especificar-lhe quais os componentes a controlar, sendo que os restantes são apenas entendidos como objetos de colisão para o planeamento de trajetórias. Surge assim o *MoveIt! Setup Assistant*, um *GUI* que permite, sumariamente:

- Analisar todo o robô e gerar uma matriz de colisões com todas as possíveis configurações das suas juntas, de modo a detetar eventuais colisões entre elas;
- Definir a que referencial se referem os pontos de interesse para o *MoveIt!*;

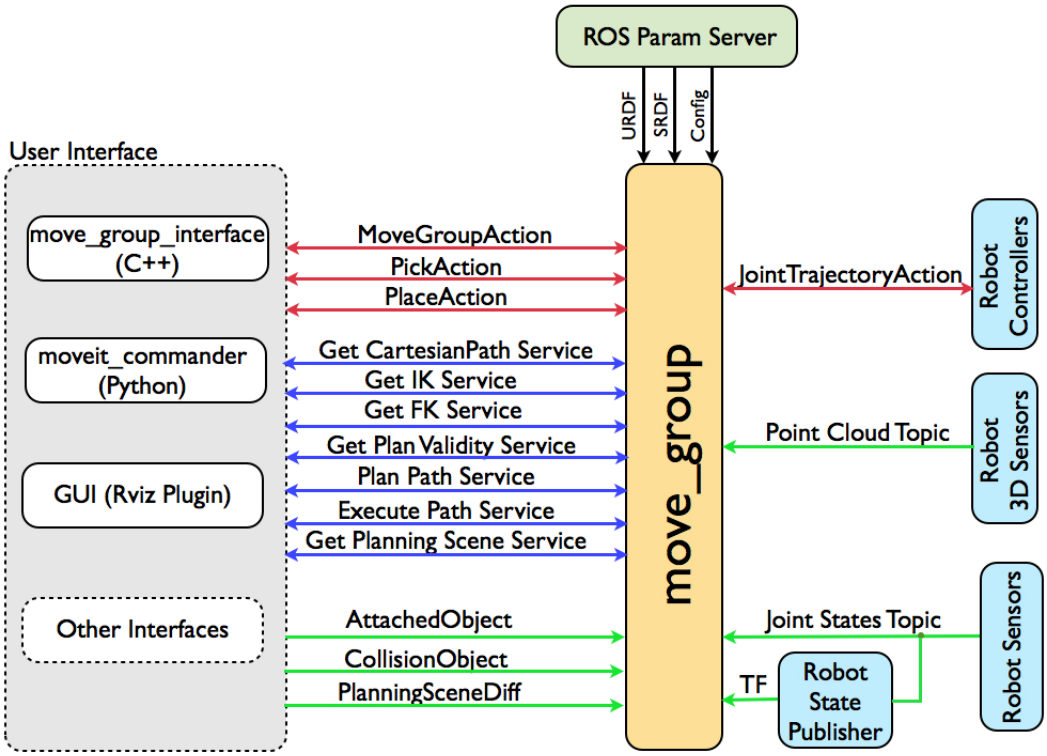


Figura 4.16: Arquitetura de sistema do *software MoveIt!*

- Selecionar grupos de planeamento de trajetórias, compostos por juntas e *end effectors*, e que podem inclusive ser operados em simultâneo;
- Definir poses *standard* dos grupos de planeamento, como por exemplo a posição *home* (original) do manipulador;
- Criar um *package* com todas estas definições e que facilmente pode ser utilizado pelo utilizador.

Esta funcionalidade do *MoveIt!* permite que facilmente o utilizador configure o seu robô com este programa, com a vantagem adicional de poder atualizar as suas configurações de forma igualmente simples. Este *plugin* comporta ainda uma vantagem adicional, implementada em Março[7] deste ano, e que foi essencial para a integração do *MoveIt!* com este robô.

Como é possível observar na figura 4.18, quando se pretende criar um *package* com as configurações do robô, é necessário selecionar o seu ficheiro base *URDF*. No entanto, robôs complexos estão divididos em vários outros robôs, através dos seus ficheiros. Assim, como forma destes robôs poderem ser acoplados a um robô base como sendo um seu componente extra, foram criados os argumentos *xacro*. Assim, com o mesmo ficheiro *launch* do *packages* do *Husky*, é possível, com chamadas específicas dos argumentos, que este robô seja lançado com um *laser* no seu topo

[7] https://answers.ros.org/question/289106/xacro-arguments-not-appearing-in-moveit-setup-assistant/

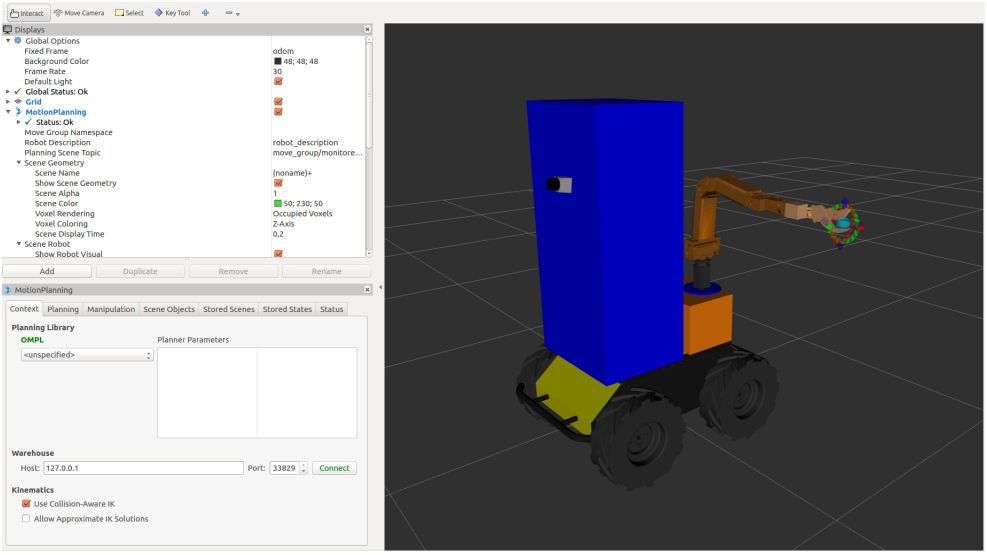


Figura 4.17: Janela do *RViz* onde o utilizador pode controlar o robô através do *GUI* desenvolvido para o *MoveIt!*

ou ainda com uma plataforma *kinect*. O facto de esta nova versão do *MoveIt!* incluir esta funcionalidade, permite lançar, através dos ficheiros base do *Husky*, o manipulador e a torre como sendo elementos adicionais à plataforma original. Caso esta funcionalidade não existisse, ainda que fosse possível simular o AGROB V16, nunca seria possível fazer com que o *MoveIt!* o reconhecesse.

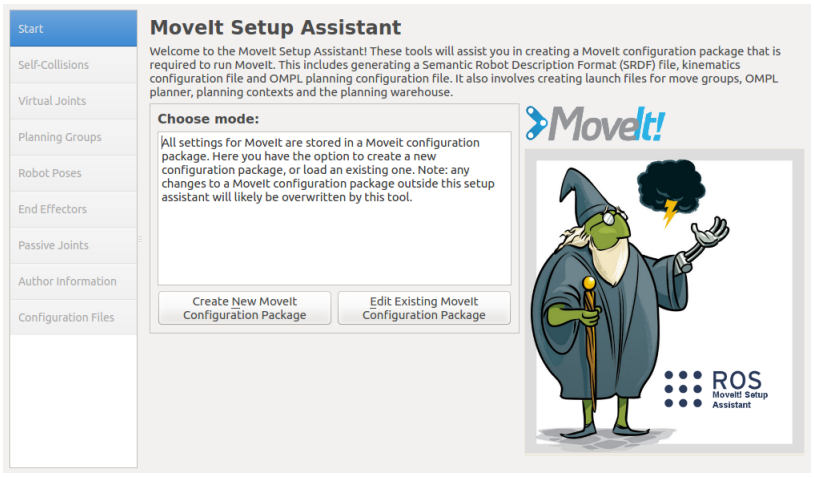


Figura 4.18: Janela do *MoveIt! Setup Assistant*

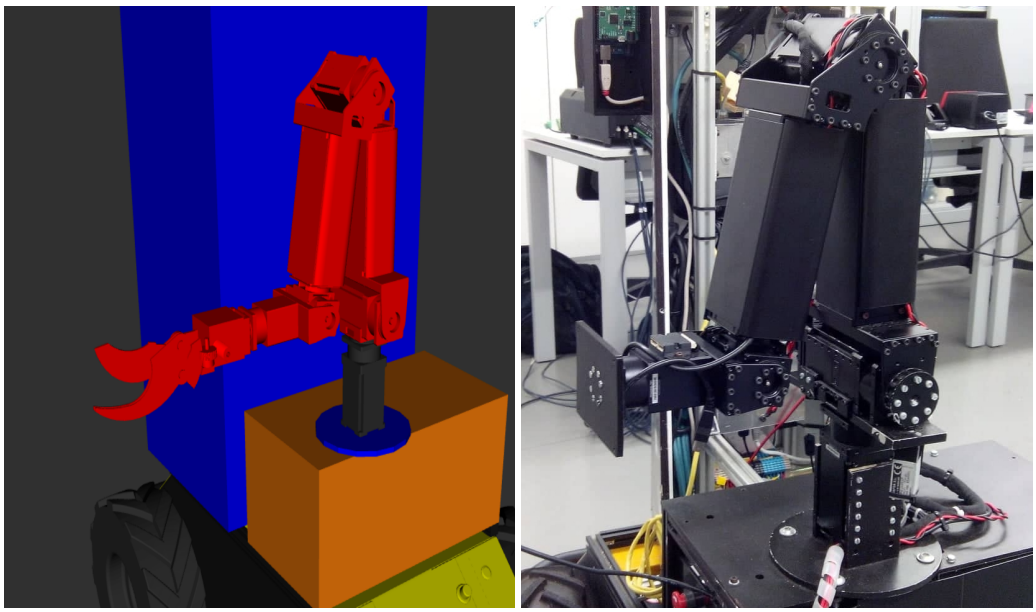
Uma vez carregado o robô neste ambiente, foi então gerada a matriz de colisões, de forma a que sejam identificadas todas as possíveis posições em que as juntas do AGROB V16 estão em colisão.

De seguida, foi selecionado o referencial ao qual se referem os pontos definidos no *MoveIt!*. Uma vez que o AGROB V16 é um robô móvel, é dotado do referencial de odometria, *odom*. Entre este referencial e o referencial *base_link* foi então definida uma junta planar (que se descola no

plano do chão), de forma a que o *MoveIt!* saiba que todos os pontos definidos se referem ao referencial *odom*.

Após este passo, foram definidas todas as juntas do AGROB V16 a controlar. Uma vez que apenas interessa controlar o braço robótico nesta plataforma, foram selecionadas todas as suas juntas, ou seja, desde o *link1* até ao *cutting_tool*, e foi dado o nome de *arm_group* a este grupo.

Como é natural que o manipulador tenha uma posição *home*, a mesma foi definida tendo em conta as especificações das posições das juntas dadas pelo co-orientador Filipe Neves dos Santos. A figura 4.19 apresenta duas imagens do manipulador, sendo a da esquerda retirada do *MoveIt! Setup Assistant*, aquando da sua configuração, e a da direita retirada do manipulador real (sem a tesoura de poda).



(a) (b)

Figura 4.19: Posição *home* do manipulador no a) *MoveIt! Setup Assistant* e no b) manipulador real (sem a tesoura de poda)

Finalmente, foi definido o componente *cutting_tool* como sendo o *end effector* do grupo *arm_group* e foi gerado o *package husky_manipulator_h_config*, com todas as configurações definidas para operar apenas o *Robotis Manipulator-H* no AGROB V16.

Uma vez completa a configuração do robô através do *MoveIt! Setup Assistant*, é apresentada de seguida o modo de interação entre o *laser Hokuyo* e o *MoveIt!*, de modo a possibilitar a perceção do ambiente por parte do programa.

4.2.2 Perceção sensorial

A perceção sensorial do ambiente é um aspeto imprescindível em qualquer sistema robótico, permitindo que este se movimente sem chocar com nenhum objeto intransponível, evitando assim danificar quer a sua estrutura, quer o objeto em questão.

O *MoveIt!* permite a localização dinâmica de obstáculos do meio através de um mapa de colisão que é continuamente expandido através dos dados enviados por sensores que lhe estejam associados. Este mapa é formado por mensagens do tipo *sensor_msgs/PointCloud2*, que permitem mapear pontos (na forma de cubos) no espaço tridimensional através de medições efetuadas por sensores, como por exemplo, o *laser Hokuyo* que se encontra no topo da torre do AGROB V16. Assim, transformando as medidas do *laser* em mensagens do tipo *sensor_msgs/PointCloud2*, e usando uma função que consiga criar um mapa de todos os pontos mapeados ao longo do percurso do robô, é possível disponibilizar este mapa ao *MoveIt!* através de um tópico.

É necessário ainda configurar o referencial ao qual se refere o mapa de colisão, uma vez que este tem que ter uma referência no *MoveIt!* que lhe permita ter perceção da localização dos obstáculos no mundo. Dado que o AGROB V16 é um veículo que se move através da rotação das suas rodas, possui um referencial de odometria, *odom*, pelo que este mapa se encontra associado a este referencial. Para demonstrar um exemplo da expansão do mapa, foi efetuada em *Gazebo* a simulação do AGROB V16 com uma árvore de grandes dimensões, que se pretende mapear. O cenário de simulação é demonstrado na figura 4.20.

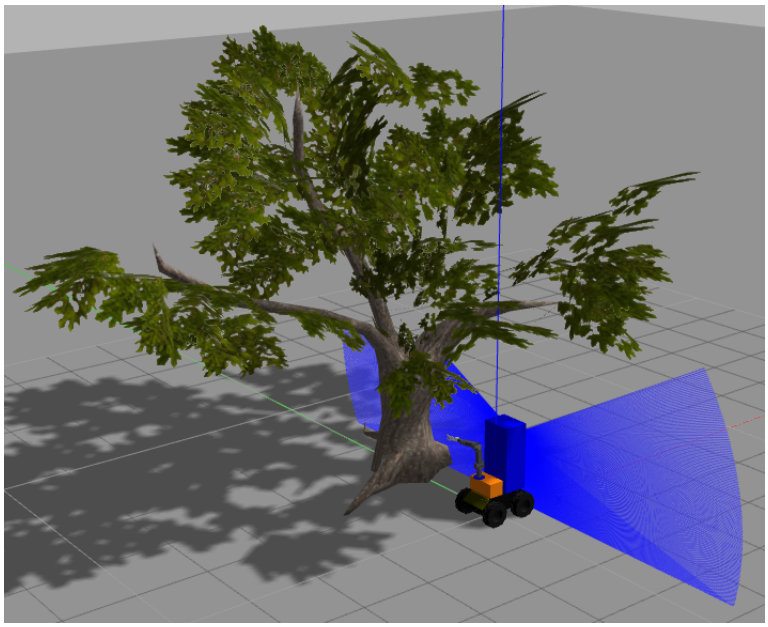
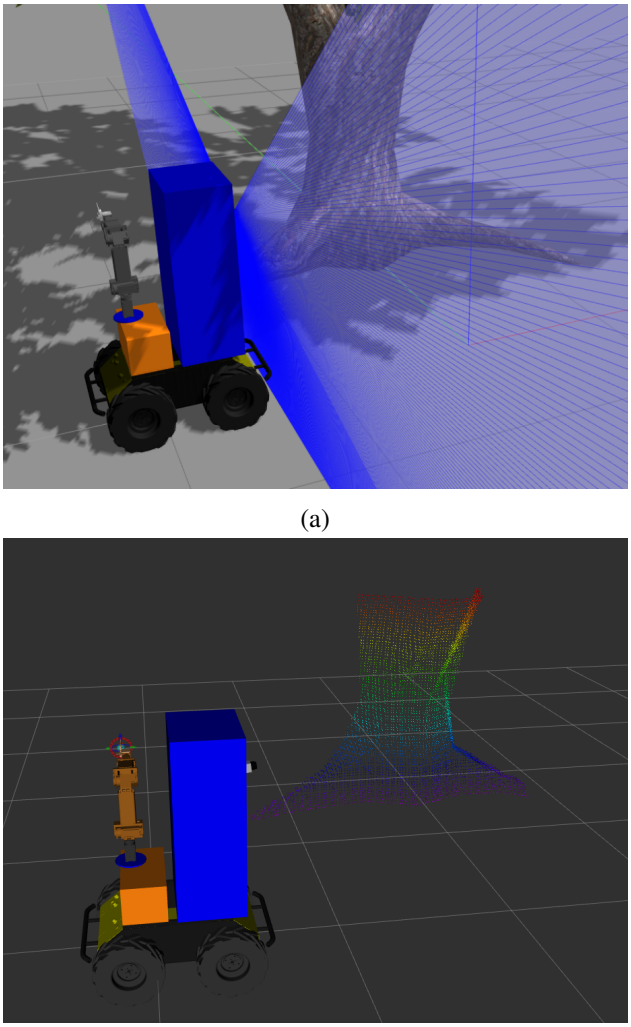


Figura 4.20: Cenário de teste para a demonstração da criação do mapa de colisão do *MoveIt!*

Como é possível observar na figura, a árvore comporta dimensões bastante superiores às do AGROB V16. No entanto, as dimensões do mapa de colisão estão limitadas pela área coberta pelo *laser Hokuyo*, que, neste cenário, cobre praticamente só o tronco da árvore. Por esta razão, apenas o tronco será mapeado.

Tal como já foi referido, o mapa de colisão é expandido através de pontos que vão sendo adicionados ao mapa e que são resultantes de diferentes medidas do sensor, originadas pelo movimento do robô. De forma a mover a plataforma, foi lançado um nó que permite telecomandar o AGROB V16 através do teclado, pelo que este foi movido até que o *laser* cobrisse toda a árvore, permitindo

assim gerar o seu mapa de colisão. A figura 4.21 apresenta a posição final do robô no *Gazebo* e no *RViz*, onde é possível verificar que o mapa de colisão que representa a árvore foi criado à medida que o robô se desloca ao longo dela.



(a)

(b)

Figura 4.21: Posição final do robô nos simuladores a) *Gazebo* e b) *RViz* após o mapeamento da árvore

De notar que a variação de cores do mapa representa a altura dos pontos mapeados, sendo que cores frias representem as zonas mais baixas do mapa, enquanto cores quentes representam as mais altas.

A densidade dos pontos do mapa também pode ser regulada, fazendo com que o detalhe do objeto mapeado seja maior ou menor. Para esta árvore, o detalhe foi alterado do seu valor original de 2,5cm para 5mm, valor otimizado para mapear árvores com troncos de menores diâmetros, como é o caso das vinhas. Como contrapartida para um maior detalhe, nem sempre é possível obter uma representação da árvore onde os pontos estão todos conectados entre si, além de que o robô se deve deslocar mais lentamente de modo a conseguir processar o maior número de pontos.

A título de exemplo, a figura 4.22 representa a mesma árvore mapeada com o detalhe original, que demonstra claramente uma representação mais fidedigna da mesma. Este parâmetro deve por isso ser calibrado consoante a aplicação para o qual se destina.

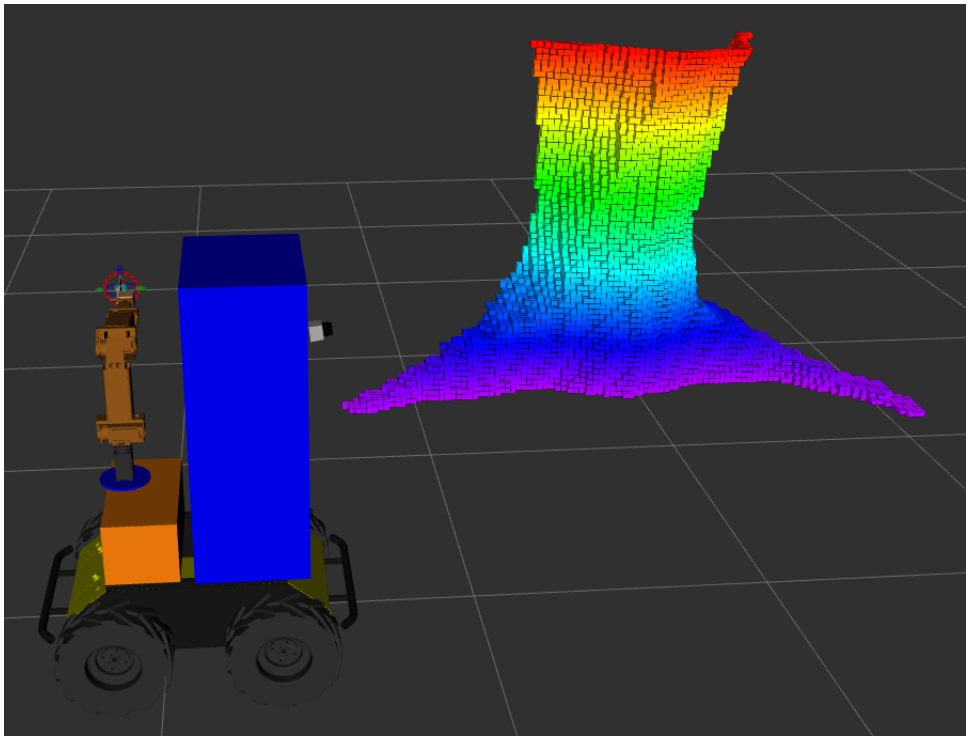


Figura 4.22: Árvore da figura 4.20 mapeada com o detalhe original do *MoveIt!*

Uma vez criado este mapa, qualquer plano que tente executar um movimento do *end effector* do robô para o interior da árvore vai falhar, dado que a árvore já foi identificada pelo *MoveIt!* como sendo um objeto intransponível de colisão.

Uma vez apresentado o modo como foi possível integrar o AGROB V16 com o *MoveIt*, são de seguida apresentadas as principais funcionalidades deste programa que se revelaram úteis na obtenção de resultados.

4.2.3 Principais funcionalidades

Ainda que seja uma ferramenta complexa, o *MoveIt!* pretende ser bastante simples de interagir, seja qual for o seu propósito. No âmbito desta dissertação, as suas principais propriedades são o facto de permitir facilmente:

- Permutar entre os algoritmos de planeamento de trajetórias;
- Planear e executar trajetórias;
- Desenhar texto e outro tipo de marcadores no *GUI* do *RViz* (para tornar testes e resultados mais intuitivos);

- Criar modelos de colisão através de modelos de objetos.

Cada uma destas quatro funcionalidades será em seguida explicada.

Escolha e configuração de planeadores

O *MoveIt!* implementa os 23 diferentes algoritmos do subcapítulo 3.2. É de esperar que o utilizador não esteja familiarizado com todos, e muito menos com as suas implementações no *MoveIt!*. Ainda que as suas implementações na *OMPL* estejam disponíveis *online*[8], o programa mantém-se o mais transparente possível.

De forma a facilitar a interação do utilizador com este *software*, apenas possível é possível alterar parâmetros que afetem o funcionamento dos algoritmos, existindo parâmetros comuns entre todos os algoritmos e outros específicos às caraterísticas particulares de cada algoritmo. Estas alterações podem ser efetuadas através do *GUI* do *MoveIt!*, disponível no *RViz*, na janela do canto inferior esquerdo (figura 4.17). Os planeadores podem ainda ser configurados no código escrito do nó que é lançado para requisitar a execução de movimentos ou outras operações.

Planear e executar trajetórias

Como já foi enunciado várias vezes ao longo do documento, o planeamento e execução de trajetórias é o principal propósito do *MoveIt!*. Após a configuração do manipulador através do *MoveIt! Setup Assistant*, os passos para executar um movimento no braço, quer em ambiente real, quer em ambiente simulado, são os seguintes:

- Escolher o grupo de planeamento para o qual vai ser efetuado o planeamento de trajetórias;
- Escolher o ponto onde se pretende posicionar o *end effector* do grupo de planeamento, tendo em conta o referencial (previamente configurado) a que se refere este ponto;
- Escolher o planeador e respetivos parâmetros que vão efetuar o planeamento de trajetórias;
- Escolher o tempo limite máximo para o cálculo da solução do planeamento de trajetórias;
- Efetuar o planeamento da trajetória;
- Caso o planeamento seja bem sucedido, executar o movimento.

Embora os passos para a execução de movimentos com o braço sejam bastante simples, existem pequenos pormenores a ter em conta.

Quando é escolhido o ponto final da trajetória, há que definir, além da sua posição, a sua orientação. *End effectors* complexos, como é o caso de uma tesoura de poda, podem nem sempre conseguir atingir o mesmo ponto em várias orientações, devido às dinâmicas que impõem ao manipulador. Por esta razão, por vezes o *MoveIt!* pode indicar que não consegue efetuar o planeamento

[8] Código fonte disponível em: https://github.com/ompl/ompl/tree/master/src/ompl

de trajetórias para um ponto que à partida é acessível, quando apenas se trata de uma questão da orientação do ponto não ser alcançável, e não a sua posição.

Outro aspeto importante é a separação entre o planeamento de trajetórias e a execução de trajetórias, que se deve essencialmente a duas razões. A primeira tem que ver com questões de segurança. Por exemplo, o utilizador pode querer efetuar o planeamento prévio de uma trajetória, mas apenas executá-la aquando da receção de um sinal. A segunda tem que ver com a variável que é criada quando é calculada com sucesso uma trajetória, que pode ser usada pelo *MoveIt! visual tools*. Esta funcionalidade é abordada de seguida.

MoveIt! visual tools

O *MoveIt! visual tools* é uma classe de objetos do *MoveIt!* que permite ao utilizador desenhar texto, pontos e outros marcadores no *GUI* do *RViz*. É bastante útil, dispondo de funções que lhe permitem, por exemplo, desenhar o ponto final de uma trajetória na janela do *RViz*, de modo a poder verificar se este se encontra na posição e orientação correta. Tal como no *MoveIt! Setup Assistant*, também esta ferramenta tem que estar associada a um referencial. Normalmente, os referenciais das duas ferramentas coincidem, sendo neste caso o referencial da odometria escolhido para ambos. Caso o utilizador queira desenhar pontos referentes a outros referenciais, pode criar outro objeto desta classe e associá-lo a um outro referencial. A figura demonstra o tipo de marcadores que foram utilizados para calibrar as posições finais dos pontos de interesse para a execução dos testes.

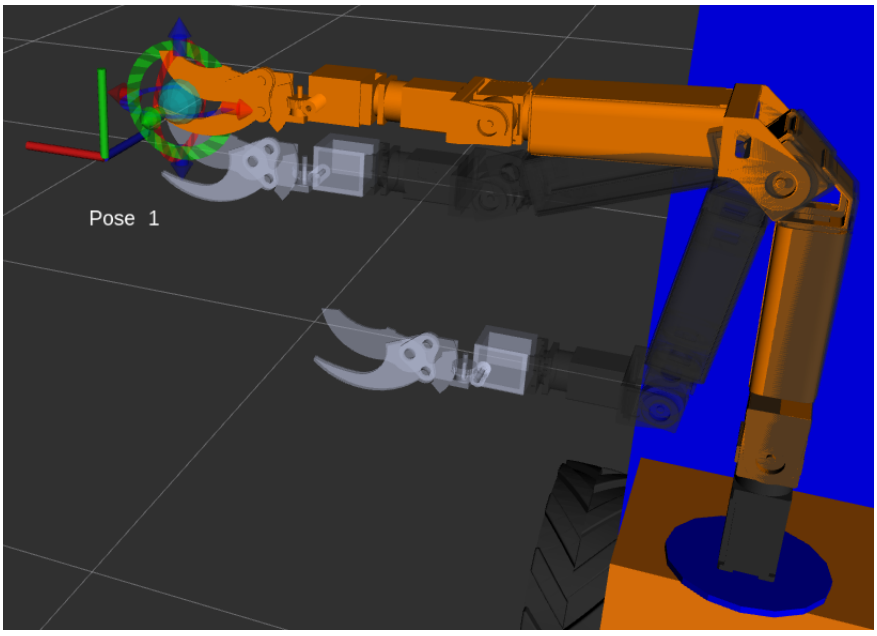


Figura 4.23: Exemplo de um dos marcadores utilizado para a calibração de pontos de interesse

Modelos de colisão

Além do desenho de marcadores recorrendo ao *MoveIt! visual tools*, o *MoveIt!* dispõe ainda de uma funcionalidade imprescindível para quem possui o modelo *URDF* do ambiente onde o robô possa operar. É comum, em ambientes controlados, serem criados modelos de teste, que têm a particularidade de poderem ser parametrizados à medida pelo utilizador, algo que nem sempre é possível num ambiente real onde vai operar um robô. Ainda assim, estes modelos são úteis para testes em ambientes controlados, especialmente no que toca a operações de *collision avoidance*, como é o caso.

No caso da operação do AGROB V16 em ambiente real, todos os seus obstáculos na vinha terão que ser mapeados com o *laser* que se encontra no seu topo. Isto supõe, pela arquitetura da plataforma, que esta tem que percorrer toda a vinha antes de se dirigir a qualquer ponto de corte nela, ponto este que eventualmente pode estar posicionado no seu início, o que suporia que a plataforma teria de se deslocar novamente até ao início da vinha, ou pelo menos até uma posição onde fosse possível executar o movimento do manipulador até esse ponto.

Num ambiente controlado, como é o caso da simulação, é possível efetuar esta mesma operação, mapeando todo o conhecido modelo da vinha como um objeto de colisão, e voltando ao ponto inicial efetuar os movimentos do braço que fossem requisitados. Para evitar que o mapeamento deste modelo de vinha, que é conhecido, seja efetuado sempre que se inicia um teste na simulação, o *MoveIt!* dispõe de uma função que lhe permite automaticamente inserir no *GUI* do *RViz* objetos de colisão conhecidos, que automaticamente serão evitados aquando do planeamento de trajetórias. Para o exemplo do modelo da árvore da figura 4.20, o modelo de colisão da árvore no *GUI* do *RViz* toma a representação da imagem da figura 4.24, sendo assim dispensável que o AGROB V16 a mapeie de cada vez que são efetuados testes na simulação.

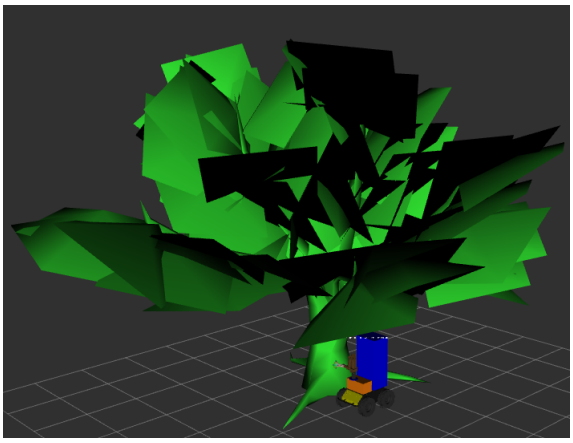


Figura 4.24: Árvore da figura 4.20 mapeada como objeto de colisão no *GUI* do *RViz*

Esta função comporta ainda uma vantagem adicional, que é o facto de, para modelos deste tipo, ser possível obter o modelo de colisão completo de todo o objeto e não só das zonas do objeto abrangidas pelo *laser*.

Uma vez apresentadas todas as informações e conhecimento necessários para um melhor entendimento do problema enunciado nesta dissertação, segue-se o capítulo de testes e resultados experimentais, onde é apresentado o *benchmarking* efetuado a todos os planeadores do *MoveIt!*, usando o modelo do AGROB V16 implementado em *Gazebo*.

Os *packages* desenvolvidos podem ser obtidos através do *link* do seguinte *link* para o *GitLab* do INESC TEC: https://gitlab.inesctec.pt/CRIIS/interwheels_arm/tree/Brito_thesis.

Capítulo 5

Testes e resultados experimentais

Neste capítulo é efetuado o *benchmarking* de todos os 23 algoritmos de planeadores da *OMPL*, implementados no *MoveIt!*, e que foram apresentados no subcapítulo 3.2. Inicialmente são apresentados os dois modelos de simulação de vinha, sendo de seguida expostas as considerações tomadas para a realização de testes nestes modelos. Segue-se a apresentação dos cinco diferentes testes efetuados no primeiro modelo e ainda de um último teste efetuado no segundo modelo, bem como a análise dos resultados obtidos. É ainda efetuada uma análise geral dos resultado e, por último, é abordada a implementação do *MoveIt!* no sistema real.

5.1 Modelos de vinhas

Os cenários de teste simulados em *Gazebo* são constituídos pelo modelo do AGROB V16, desenvolvido no capítulo 4, e ainda por dois modelos de vinha, desenvolvidos em *SolidWorks*. Estes, ainda que simples, têm como objetivo representar as vinhas da encosta do Douro, desprezando qualquer inclinação do terreno. São de seguida apresentados os dois modelos de vinha e seus propósitos.

5.1.1 Modelo de vinha simples

Como é possível observar na figura 1.3, as vinhas de encosta do Douro são plantas de estatura semelhante à do AGROB V16. Desenvolvem-se em volta de um suporte que dita a direção do seu crescimento e que consiste numa série de arames dispostos paralelamente segundo um plano aproximadamente perpendicular ao plano do chão. Uma primeira representação simples deste tipo de vinha pode considerar que os troncos se desenvolvem apenas na direção da disposição dos arames, e que estes mesmos troncos são as vides que se pretendem podar. Tendo em conta estas considerações, foi desenvolvido o primeiro modelo de vinha, apresentado na figura 5.1. De notar que é desprovido de folhas, uma vez que, na altura da poda, a vinha se encontra completamente despida.

Este modelo consiste numa pequena árvore com 1,14 metros de altura e 1,1 de largura, cujos 13 ramos se encontram dispostos no mesmo plano e separados de forma não equidistante. Os três

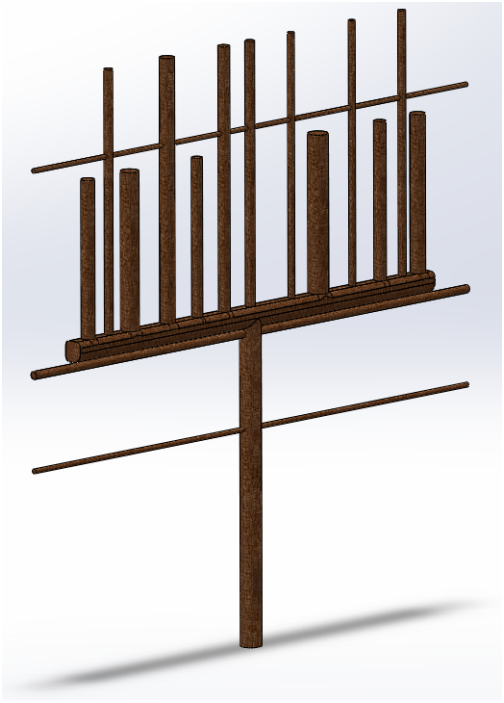


Figura 5.1: Modelo de vinha simples desenvolvido em *SolidWorks*

troncos que se encontram dispostos paralelamente ao chão simulam os arames da estrutura, por onde também podem crescer outros ramos. O diâmetro dos ramos deste modelo variam entre 1 a 2*cm*, sendo que a tesoura de poda apenas consegue cortar os troncos com grossuras inferiores à sua abertura máxima de 1,5*cm*. Desta forma, apenas foram selecionados pontos de corte na árvore passíveis de serem alcançados pela tesoura.

Ainda que esta representação careça de algum detalhe, permite simular troncos bastante próximos entre si e com diferentes diâmetros. Com vista a melhorar o detalhe do modelo, foi desenvolvido o modelo de vinha complexo.

5.1.2 Modelo de vinha complexo

O modelo de vinha complexo tem como base o modelo de vinha simples, ao qual foram adicionados novos troncos, dispostos em direções aleatórias. Estes troncos têm origem nos do primeiro modelo e tem como objetivo simular as disposições irregulares das vides que se encontram numa vinha deste tipo, que muitas vezes se sobrepõem e que são de difícil acesso ou que tornam difícil o acesso a outras vides.

Foram então adicionados 4 novos troncos ao modelo anterior. O novo modelo apresenta o aspeto das imagens da figura 5.2.

Em seguida são apresentadas as considerações que foram tomadas para a realização destes testes.

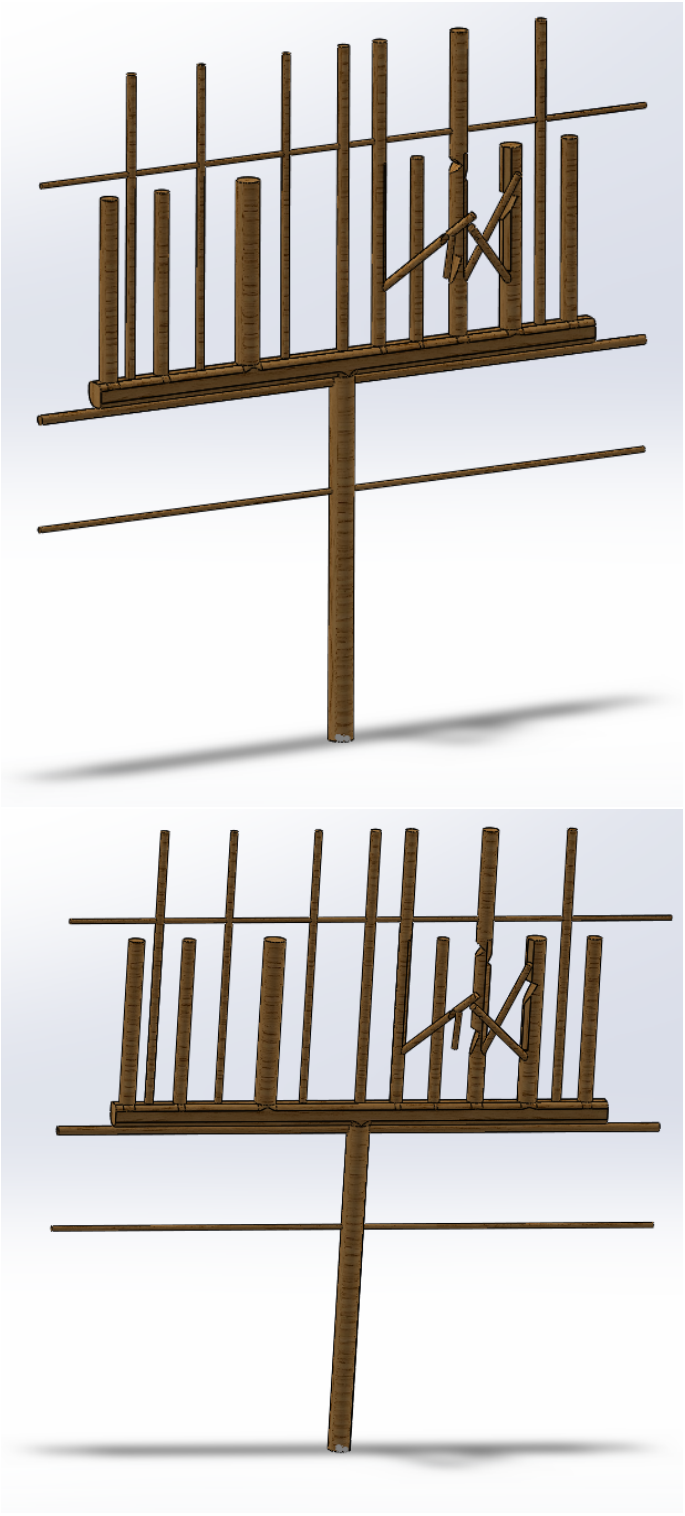


Figura 5.2: Modelo de vinha complexa desenvolvido em *SolidWorks*

5.2 Considerações de testes

Como já foi referido anteriormente, o *benchmarking* dos planeadores consiste na realização de testes em cenários simulados em *Gazebo*, onde o manipulador da plataforma AGROB V16 tem como objetivo alcançar pontos de corte nos dois modelos de vinha desenvolvidos, descrevendo trajetórias calculadas por diferentes algoritmos. O planeamento dos algoritmos deve ser sempre executado, ou da posição *home* do manipulador para a posição final de corte, ou vice-versa. Nestas situações, os tempos de planeamento são adquiridos com o objetivo principal de aferir a configuração que mais rapidamente os calcula. De notar que o tempo de execução da trajetória não é um parâmetro a avaliar, uma vez que apenas depende dos controladores dos motores, que não são objeto de estudo nesta dissertação.

Existem dois cenários de teste, sendo que as únicas diferenças entre eles são o modelo de vinha utilizado e ainda a posição desse modelo no mapa. A apresentação dos cenários é detalhada com mais pormenor nos subcapítulos 5.3 e 5.4.

São de seguida apresentados os passos e considerações comuns aos dois modelos de vinha e aos dois cenários de simulação.

5.2.1 Perceção da vinha

Em ambos os cenários de teste, e antes de se iniciar o planeamento de movimentos, o *MoveIt!* tem que ter perceção de toda a vinha de modo a poder planear tendo em conta a sua estrutura. A descrição deste passo, tanto em ambiente real como em ambiente simulado, é apresentada no subcapítulo 4.2.2, pelo que o robô tem que percorrer lentamente a vinha de modo a mapeá-la como objeto de colisão. A figura 5.3 mostra o objeto de colisão resultante do mapeamento do primeiro modelo de vinha.

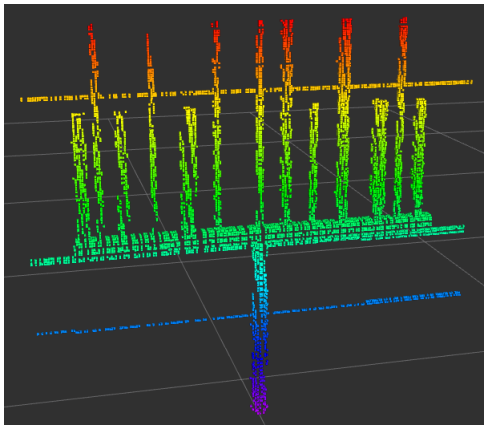
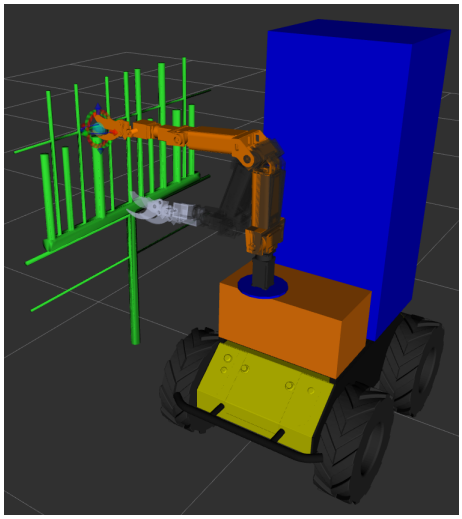


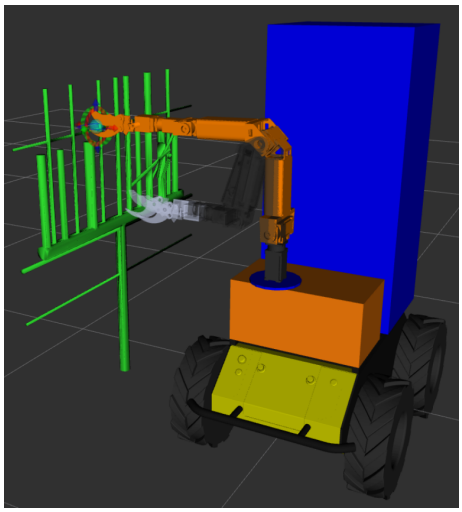
Figura 5.3: Modelo de vinha simples mapeada como objeto de colisão

Acontece que nem todos os testes foram bem sucedidos, isto é, por vezes a tesoura chocou com o modelo de vinha que, na forma como está simulada, cai com a força do impacto, ou roda sobre si mesma caso o impacto seja de raspão. Nestes casos, é necessário reiniciar a simulação, uma

vez que a árvore já não se encontra na sua posição inicial. Isto por sua vez implica que a árvore seja novamente mapeada pelo AGROB V16, o que despende algum tempo e se torna exaustivo ao fim de poucos testes. Para contornar esta particularidade, e uma vez que é conhecido o modelo da vinha, os modelos de colisão 4.2.3 do *MoveIt!* permitem que todo o modelo da vinha seja automaticamente reconhecido como um modelo de colisão, o que torna desnecessário mapear a vinha e acelera o processo de *benchmarking*. Por esta razão, ambos os modelos de vinha foram processados através desta ferramenta e não com o *laser* no topo da plataforma. Os modelos de vinha processados como modelos de colisão, encontram-se na figura 5.4.



(a)



(b)

Figura 5.4: a) Primeiro e b) segundo modelos de vinha identificados como objetos de colisão pelo *MoveIt!*

Ainda que estes modelos sejam mais precisos que o modelo obtido pelo *laser*, os resultados obtidos são igualmente fiáveis, uma vez que há bastante detalhe no modelo obtido na figura 5.4. O pormenor pode ainda ser aumentado, consoante o desempenho do processador montado na

plataforma. E como este aspeto também é importante, as caraterísticas da plataforma onde foram realizados os testes apresentam-se em seguida.

5.2.2 Plataforma de testes

Toda a dissertação, que inclui o processo de investigação, desenvolvimento e escrita, foi desenvolvida no meu portátil pessoal, o modelo *TECRA A50-C-201*[1] da *Toshiba*. As suas principais caraterísticas são o processador *i7* de 6ª geração da *Intel*, com um *clock speed* de 2.50/3.10 *Turbo GHz* e uma *cache* de nível 3 de 4*Mb*, o disco *Solid State Drive* (*SSD*), 16*Gbs* de *RAM* e ainda a placa gráfica *930MX* da *NVIDIA*.

Por último, são apresentadas as caraterísticas relativas aos algoritmos de planeamento de movimentos e aos resultados obtidos.

5.2.3 Algoritmos e resultados

Dado que que existem 36 diferentes parâmetros entre todos os 23 algoritmos, existe uma infinidade de combinações que poderiam ser testadas numa também vasta diversidade de testes que poderiam comportar várias situações particulares. Existe, no entanto, apenas um parâmetro comum a todos os planeadores e que pode ser alterado, o *longest_valid_segment_fraction* (*lvsf*), que define a distância a ser considerada entre *waypoints* de uma trajetória. Nesta aplicação, este parâmetro é determinante na computação de trajetórias com grande detalhe, como aquele que é exigido nas zonas mais próximas dos pontos de corte. Nestas zonas, a abundância de pequenos ramos é maior, o que por sua vez requer um maior pormenor nas trajetórias calculadas de modo a que o manipulador não choque com eles. Devido às limitações temporais para realizar a presente dissertação, e também de modo a facilitar a análise comparativa de todos os planeadores para esta aplicação específica, optou-se por apenas alterar este parâmetro nos demais testes realizados. Os valores que lhe foram atribuídos são indicados em cada teste específico.

Resultados fidedignos exigem um número de testes bastante extenso, pelo que não foi possível inserir todos os seus valores neste documento. Assim, apenas é apresentada a média e o valor máximo e mínimo dos tempos adquiridos em cada movimento do planeamento de trajetórias, considerando a soma dos tempos das parcelas que serão de seguidas indicadas. A apresentação destes resultados está condicionada pelo número de testes bem sucedidos, sendo este critério definido em cada teste e também apresentado nos melhores casos. Ainda assim, eventuais valores temporais de maus resultados poderão ser comparados com os obtidos nos resultados considerados positivos. As tabelas com os valores completos dos resultados podem ser consultadas no documento *LibreOffice* anexado a esta dissertação.

Para testes que consistem apenas no movimento da posição *home* do manipulador para a posição final de corte, e vice-versa, são devolvidos quatro valores de tempo que foram registados nas células do documento. O primeiro valor de cada célula corresponde ao tempo que o planeador demora a encontrar a solução desde a posição *home* para a posição final de corte. O segundo valor

[1] http://www.toshiba.pt/laptops/tecra/tecra-a50-c/tecra-a50-c-201/

é separado do primeiro com o símbolo +, e simboliza o tempo que o algoritmo demora a simplificar o caminho gerado pela solução. Só após estes dois tempos é que o manipulador se começa efetivamente a mover e a percorrer a trajetória planeada. Uma vez atingido o ponto pretendido, os dois primeiros valores são separados dos segundos através da palavra *and*, e é iniciado novamente o algoritmo, sendo que os restantes dois tempos são registados da mesma maneira.

De notar ainda que as implementações dos algoritmos *LBTRRT*, *LazyPRM*, *LazyPRMstar*, *PRM*, *PRMstar*, *RRTstar*, *SPARS* e *SPARStwo* têm em comum o facto de despenderem sempre o tempo máximo imposto para encontrarem a solução do algoritmo, cujo valor é indicado em cada teste. As vantagens deste processo são: (1) não existe etapa de simplificação do caminho gerado pela solução (2) o facto de ser possível controlar estritamente o tempo despendido no planeamento. A desvantagem é o tempo necessário para a obtenção de bons resultados possa ser demasiado alto, o que torne a solução lenta e consequentemente inviável. Os valores deste limite temporal são indicados em cada teste, sendo que, no caso de um planeamento bem sucedido, é apresentado um *S* que simboliza este valor na vez dos dois valores indicados para cada movimento.

Foram obtidos cinco tipos de resultados durante os testes realizados: (1) trajetória efetuada com sucesso, (2) trajetória colide com a vinha, (3) não foi possível calcular a trajetória, (4) não foi possível atingir o ponto e (5) o nó do *MoveIt!* encerrou subitamente. A situação em que o ponto não é atingido verifica-se quando, a meio de um trajetória, o *MoveIt!* indica que atingiu o ponto de destino e começa a efetuar novo planeamento. A causa para o de encerramento súbito do nó não foi identificada, visto que acontecia espontaneamente e apenas em 2 algoritmos. Nas tabelas apresentadas, as células que correspondem ao primeiro resultado estão coloridas com fundo branco, as seguintes com fundo vermelho, bordô, laranja e rosa, respetivamente. Não foi feita distinção de cores quando a trajetória colide com a vinha e esta cai (*bump*), ou quando o toque é apenas de raspão e esta gira sobre si própria (*small bump*). É apenas feita a distinção entre a colisão no primeiro ou segundo movimentos, consoante aparecerem quatro ou dois números, respetivamente.

Por último, na apresentação de testes em tabelas, as abreviaturas da tabela 5.1 foram utilizadas para a legendagem das colunas.

Termo	Abreviatura
Tempo máximo	Tmax
Tempo mínimo	Tmin
Tempo médio	Tmed
Tempo médio por ponto	TmedP
Número de sucessos	Nsucessos
Número de total de sucessos no teste	Nsucessos total

Tabela 5.1: Abreviaturas usadas nas tabelas de resultados

O código implementado para a realização dos testes, juntamente com as linhas de código a digitar na linha de comandos para executar a simulação em *Gazebo*, o nó do *MoveIt!* e o nó que requisita as trajetórias do manipulador e que envia comandos de velocidade ao *Husky*, podem ser

acedidos através do seguinte *link* para o *GitLab* do INESC TEC:

https://gitlab.inesctec.pt/CRIIS/interwheels_arm/tree/Brito_thesis.

Em seguida são apresentados os cenários de teste, os testes neles realizados e respetivos resultados.

5.3 Modelo AGROB V16 e vinha simples

O primeiro cenário de testes é constituído pelo modelo do AGROB V16 e da vinha simples, ambos simulados em *Gazebo*. A plataforma AGROB V16 começa todos os testes na posição (x,y,z) = (0, 0, 0) e a vinha encontra-se a 80cm da plataforma, na direção positiva do eixo dos yy. A figura 5.5 apresenta uma imagem deste cenário no *Gazebo*.

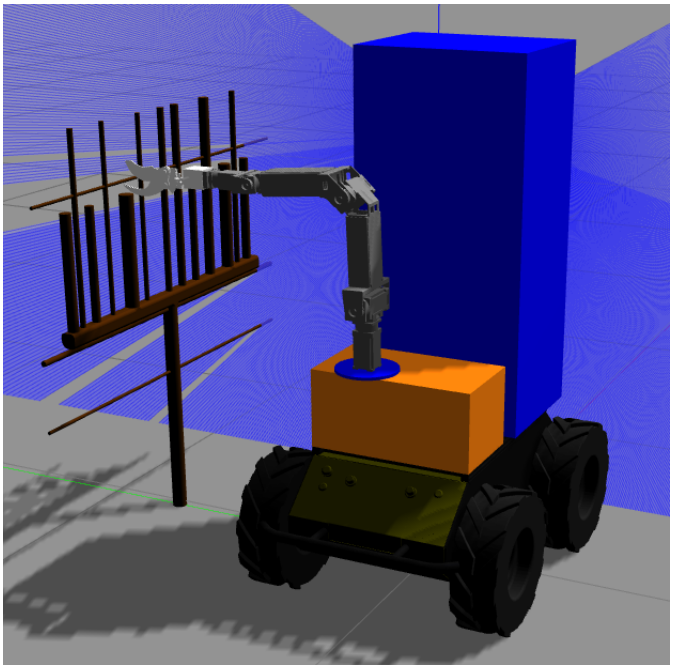
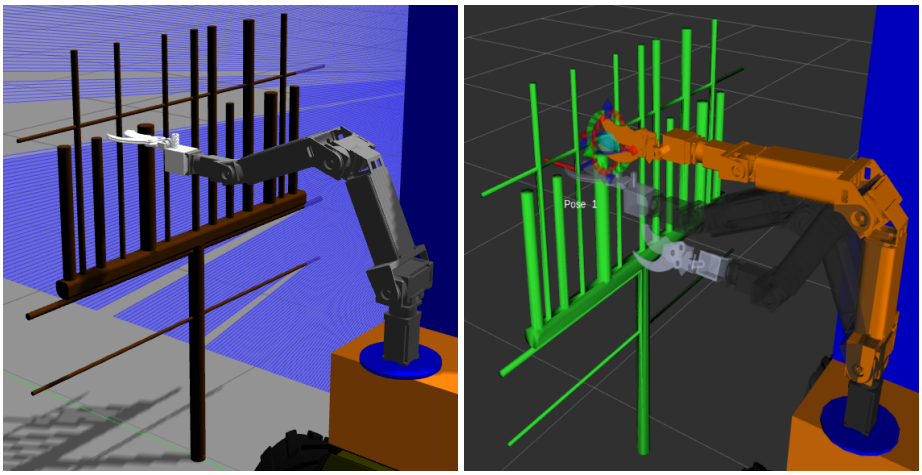


Figura 5.5: Primeiro cenário de testes - *Gazebo*

Foram realizados 5 testes. Os primeiros 4 consistiam em levar o manipulador da sua posição *home* para uma posição final de corte, e voltar de novo para a posição inicial. Na figura 5.6 é apresentada a posição final de corte.

O mesmo algoritmo foi usado nos dois movimentos, de forma a registar separadamente o seu comportamento nas duas situações. Para cada algoritmo foram efetuadas 20 tentativas, o que totaliza um total de 40 trajetórias, número que se considerou suficiente para a obtenção de resultados fidedignos.

O último teste neste cenário destina-se aos planeadores que apresentaram melhores resultados nos dois movimentos acima descritos. Para a sua escolha, inicialmente é formado um grupo com os algoritmos que, no máximo, não foram capazes de planear corretamente os dois movimentos um máximo de três vezes. Dentro deste conjunto, por sua vez são estudados individualmente os



(a) (b)

Figura 5.6: a) Posição final de corte no *Gazebo* para os quatros primeiros testes e b) marcador desta posição no *RViz*

dois movimentos, de forma a que as suas configurações possam ser combinadas para a obtenção de melhores resultados nos mesmos dois movimentos em 5 pontos espalhados pela vinha. A figura 5.7 apresenta os 5 pontos de corte deste quinto teste. De notar que o eixo vermelho representa a direção na qual aponta a tesoura, o eixo azul é coincidente com o plano que interceta as suas faces e o eixo verde, quando se encontra no 1º ou 2º quadrante do plano que o interseta a ele e ao eixo azul, indica que as lâminas da tesoura apontam para cima (posição normal).

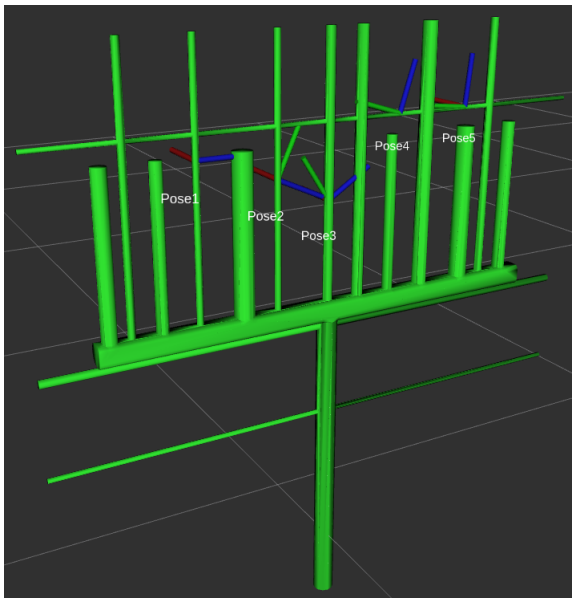


Figura 5.7: 5 pontos de corte do quinto teste do primeiro cenário de testes

Foram efetuados 10 testes para cada combinação, o que totaliza 50 trajetórias calculadas. Nesta situação, o AGROB V16 desloca-se ao longo da vinha de cada vez que um plano para o

primeiro movimento não encontre solução, pelo que a velocidade com que este se desloca ao longo da vinha está dependente do tempo máximo atribuído ao planeamento. Poderiam ainda ser impostas outras regras heurísticas, como posicionar o referencial *base_link* do *Husky* a menos de uma certa distância (*offset*) em *xx* dos pontos pretendidos. Ainda assim, a primeira regra foi preferida, de modo a analisar o comportamento do *MoveIt!* quando o manipulador nem sempre se encontra perto dos pontos de corte.

5.3.1 Teste com o valor de 5*cm* no parâmetro *lvsf*

No primeiro teste do primeiro cenário são testados todos os algoritmos configurados com o valor por defeito do parâmetro *longest_valid_segment_fraction*: 5*cm*. Para os planeadores não limitados em tempo, o valor máximo definido para o planeamento é de 1 minuto, enquanto que os planeadores que estão limitados temporalmente apenas dispõem de 1 segundo. Os resultados podem ser consultados nas linhas 8 a 27 das tabelas anexadas a esta dissertação, em baixo da célula *TEST 1*, preenchida a azul claro. Os dados da tabela 5.2 apresentam os mais relevantes resultados que são de seguida analisados.

Nº de algoritmos cujas trajetórias planeadas entraram em colisão com a vinha pelo menos 10 vezes	19
Nº de algoritmos que conseguiram planear trajetórias livres de colisões 10 ou mais vezes	4
Nº de algoritmos cujos planos colidem com a vinha um máximo de 3 vezes	0
Nº de algoritmos que não encontraram solução pelo menos 10 vezes	6
Nº de algoritmos cujas trajetórias calculadas colidiam com a árvore 10 ou mais vezes	8
Nº de algoritmos que nunca encontraram solução	2
Nº de algoritmos que calcularam trajetórias não naturais	9
Nº total de colisões com a vinha	164
Nº de colisões no primeiro movimento	101
Nº de colisões no segundo movimento	63
Nº total de soluções não encontradas	115
Nº de solução não encontradas no primeiro movimento	109
Nº de solução não encontradas no segundo movimento	6
Nº de algoritmos onde se verificou o encerramento súbito do nó do *MoveIt!*	2

Tabela 5.2: Sumário de resultados do primeiro teste

Nas tabelas anexadas observa-se uma grande abundância da cor vermelha e bordô, o que revela que a maior parte dos algoritmos calculou trajetórias que colidem com a vinha, ou que simplesmente não foram capazes de calcular nenhuma trajetória. Os algoritmos *SPARS* e *SPARStwo* são exemplo deste caso pois nunca foram capazes de encontrar solução.

Dos 23 algoritmos, 19 não foram capazes de calcular trajetórias que não entrassem em colisão com a vinha pelo menos 10 vezes, sendo que apenas 4 o conseguiram 10 ou mais vezes. Os melhores resultados foram obtidos pelos algoritmos *RRTConnect* e *SBL*, que conseguiram planear 12 vezes corretamente, não sendo selecionados, nesta configuração, para o quinto teste. O melhor resultado obtido pelos algoritmos limitados em termos temporais foi o do *LazyPRMstar*, que calculou corretamente 10 trajetórias. Os piores resultados são os dos algoritmos *BFMT*, *BiEST*, *EST*, *LBTRRT*, *PRM*, *RRT*, *RRTstar*, *SPARS*, *SPARStwo* e *TRRT*, que conseguiram no máximo planear corretamente 3 vezes.

Os algoritmos *LBTRRT*, *RRT*, *RRTstar*, *SPARS*, *SPARStwo* e *TRRT* não encontraram solução pelo menos 10 vezes e os algoritmos *BKPIECE*, *BiEST*, *BiTRRT*, *EST*, *FMT*, *PRM*, *PRMstar* e *STRIDE* bateram na árvore pelo menos 10 vezes. Todos os impactos registados fizeram com que a vinha tombasse, e isso deve-se, em parte, a algumas trajetórias computadas que faziam o manipulador descrever movimentos não naturais, isto é, movimentos para posições que não representam a trajetória naturalmente esperada. A figura 5.8 demonstra um exemplo de uma das posições do robô enquanto percorre uma trajetória deste tipo.

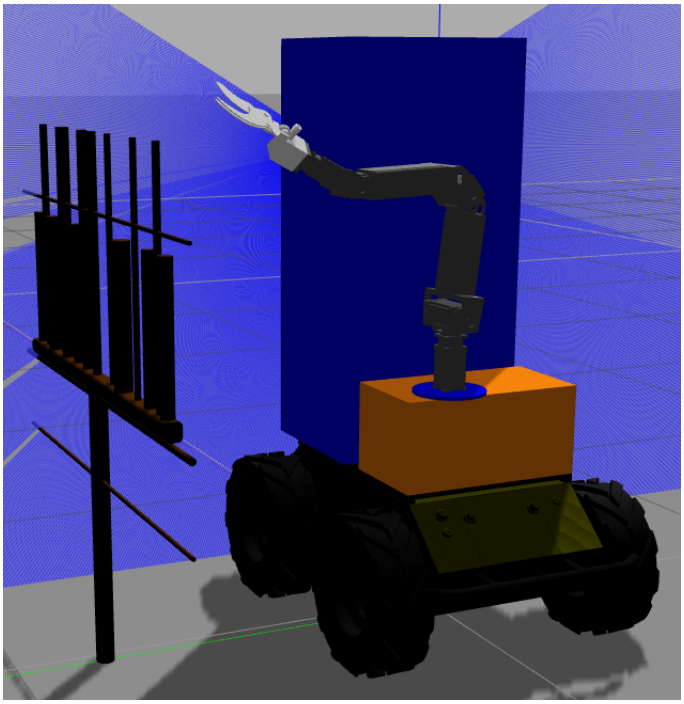


Figura 5.8: Exemplo de uma posição do manipulador enquanto percorre uma trajetória dita estranha

Como o manipulador se encontra bastante próximo do ponto de corte, não é de esperar que tenha de atingir a posição da figura antes de se dirigir a ele. Quando trajetórias deste tipo foram calculadas, situação que se verificou mais recorrentemente nos algoritmos *KPIECE* (8) e *LBKPIECE* (5), o manipulador ora chocava com a vinha ora não conseguia atingir o ponto de corte. No total, foram calculadas trajetórias não naturais em 9 algoritmos.

Ainda relativamente aos impactos, foram totalizados 164, sendo que 101 deles ocorreram no primeiro movimento, enquanto apenas 63 ocorreram no segundo, o que indicia que este aparenta ser mais fácil de computar. O número total de vezes que houve falha no planeamento de movimentos é de 115, sendo que 109 ocorreram no primeiro movimento e apenas 6 no segundo. O facto do número de incidências ser sempre menor no segundo caso pode dever-se à premissa de que apenas existe um segundo movimento quando o primeiro é bem sucedido, o que resulta num muito menor número de segundos movimentos do que primeiros.

Apenas nos algoritmos *BFMT* e *LBTRRT* ocorreram situações de encerramento súbito do nó, situação que ocorreu 15 vezes no primeiro caso e 3 vezes no segundo.

Por último, os dois algoritmos com melhor desempenho neste teste apresentaram os resultados que se encontram nas tabelas 5.3 e 5.4, para o primeiro e segundo movimentos, respetivamente.

	Movimento da posição home para o ponto de corte				Nsucessos
	Tmax (s)	Tmin (s)	Tmedio (s)	Nsucessos	total
RRTConnect	0,504	0,167	0,344	16/20	12/20
SBL	0,721	0,33	0,495	16/20	12/20

Tabela 5.3: Tabela com os melhores resultados do 1º movimento do 1º teste do 1º cenário

	Movimento do ponto de corte para a posição home				Nsucessos
	Tmax (s)	Tmin (s)	Tmed(s)	Nsucessos	total
RRTConnect	0,433	0,135	0,218	12/16	12/20
SBL	0,774	0,193	0,382	12/16	12/20

Tabela 5.4: Tabela com os melhores resultados do 2º movimento do 1º teste do 1º cenário

Analisando os resultados das tabelas é possível observar que, apesar de os testes só terem sido completados 12 vezes em ambos os casos, os movimentos individuais obtêm resultados consideravelmente melhores, ainda que não tenham atingido a meta estipulada. É possível ainda começar a ter uma perceção do tempo computacional exigido para o cálculo de trajetórias neste tipo de ambientes, sendo que, em média, não são esperados valores abaixo dos 0,344s para o primeiro movimento e de 0,218s no segundo movimento. Ainda assim, há que ter em conta que estes valores são obtidos para resultados desfavoráveis de trajetórias calculadas com sucesso, pelo que é de esperar que ambos subam em troca de uma melhor performance no cálculo de trajetórias.

De modo geral todos os algoritmos apresentaram resultados bastante fracos, pelo que com este valor do parâmetro *longest_valid_segment_fraction* nenhuma configuração poderia ser implementada na plataforma real, correndo-se o risco de danificar quer as vinhas, quer a plataforma, além de que o robô teria um baixo rendimento pois no máximo podaria cerca de 60% das vinhas, valor bastante baixo.

É de seguida apresentado o segundo teste do primeiro cenário, onde o valor do parâmetro é alterado.

5.3.2 Teste com o valor de 1*cm* no parâmetro *lvsf*

No segundo teste do primeiro cenário todos os algoritmos são novamente testados mas desta vez com o valor de 1*cm* no parâmetro *longest_valid_segment_fraction*. Para os planeadores não limitados em tempo, o valor máximo definido para o planeamento é agora de 1 minuto, enquanto que os planeadores que estão limitados temporalmente apenas dispõem de 2 segundos. Os resultados podem ser consultados nas linhas 38 a 57 das tabelas que se encontram anexadas a esta dissertação, em baixo da célula *TEST 2*, preenchida a azul claro. Os dados da tabela 5.5 apresentam os mais relevantes resultados que são de seguida analisados.

Nº de algoritmos que planearam trajetórias que entram em colisão com a vinha no máximo 6 vezes	11
Nº de algoritmos que conseguiram planear trajetórias livres de colisões 14 ou mais vezes	12
Nº de algoritmos cujas trajetórias calculadas colidem com a vinha no máximo 3 vezes	6
Nº de algoritmos que não encontraram solução pelo menos 14 vezes	10
Nº de algoritmos que nunca encontraram solução	5
Nº de algoritmos cujas trajetórias calculadas colidiam com a árvore 10 ou mais vezes	0
Nº de algoritmos que calcularam trajetórias não naturais	0
Nº total de colisões com a vinha	37
Nº de colisões no primeiro movimento	24
Nº de colisões no segundo movimento	13
Nº total de soluções não encontradas	183
Nº de solução não encontradas no primeiro movimento	161
Nº de solução não encontradas no segundo movimento	22
Nº de algoritmos onde se verificou o encerramento súbito do nó do *MoveIt!*	1

Tabela 5.5: Sumário de resultados do segundo teste

A primeira grande diferença observada nas tabelas anexadas é que há agora muito mais células em branco, fruto dos melhores resultados obtidos, que são um primeiro indicador de que este parâmetro é de facto impactante.

Dos 23 algoritmos, 12 calcularam trajetórias que não entram em colisão com a vinha pelo menos 14 vezes, sendo que os algoritmos *BKPIECE*, *BiEST*, *EST*, *KPIECE*, *LBKPIECE* e *LazyPRMstar* falharam no máximo 3 vezes, pelo que foram selecionados, com esta configuração do parâmetro *longest_valid_segment_fraction*, para o grupo de planeadores que poderá estar incluído no quinto reste. Entre todos estes seis algoritmos, apenas o *LazyPRMstar* é limitado temporalmente e os algoritmos *EST* e *KPIECE* tiveram desempenhos perfeitos em termos de assertividade, não tendo colidido nenhuma vez com a vinha. Os piores resultados são os dos algoritmos *BFMT*, *BiTRRT*, *LBTRRT*, *LazyPRM*, *PRM*, *PRMstar*, *RRT*, *RRTstar SPARS*, *SPARStwo* e *TRRT*, que conseguiram no máximo planear corretamente 6 vezes.

Aos algoritmos *LBTRRT*, *RRT*, *RRTstar*, *SPARS*, *SPARStwo* e *TRRT* que não tinham encontrado solução em pelo menos 10 no primeiro teste, juntaram-se os algoritmos *BiTRRT*, *LazyPRM*, *PRM* e *PRMstar* sendo que este valor piorou para 14. Ainda nesta situação, aos algoritmos *SPARS* e *SPARStwo*, que não encontraram nenhuma solução no primeiro teste, juntou-se o *LBTRRT*, o *RRT* e *RRTstar* neste teste. No total ocorreram 183 situações deste tipo, havendo um aumento do seu número quer no primeiro movimento quer no segundo, que agora totalizam os valores de 161 e 22, respetivamente. Outra diferença notória é o facto do número de impactos com a vinha, quer no primeiro quer no segundo movimento, diminuíram, sem qualquer exceção. Os novos números são agora de 24 impactos no primeiro movimento contra 13 no segundo. Além disso, nunca mais foi verificada nenhuma trajetória não natural, e os impactos passaram quase todos a ser de raspão, tendo-se apenas verificado a queda da vinha 4 vezes, 1 no primeiro movimento e 3 no segundo.

As situações de encerramento súbito do nó passaram agora a verificar-se apenas no algoritmo *BFMT*.

15 algoritmos melhoraram o seu desempenho, segundo os valores descritos na tabela 5.6. Os restantes pioraram os seus desempenhos, exceto nos casos mencionados dos algoritmos que novamente não foram capazes de planear corretamente uma única vez, onde o resultado se manteve nulo.

	1º teste	2º teste
BFMT	3	5
BKPIECE	7	18
BiEST	5	19
EST	3	19
FMT	8	14
KPIECE	8	20
LBKPIECE	8	18
LazyPRMstar	10	17
PDST	11	15
PRM	1	4
PRMstar	5	6
ProjEST	10	16
RRTConnect	12	16
SBL	12	16
STRIDE	8	15

Tabela 5.6: Algoritmos e valores de sucesso que melhoraram do 1º para o 2º teste

Estes dados são assim mais um indicador da melhoria de performance que pode ser alcançada através da alteração do parâmetro em questão. De notar ainda as melhorias drásticas de assertividade dos 6 algoritmos que foram selecionados para o 5º teste, cujas valores ultrapassaram aqueles dos melhores dois planeadores do primeiro teste. Os seus dados para este teste encontram-se nas tabelas 5.7 e 5.8, para o primeiro e segundo movimentos, respetivamente.

Analisando os resultados da tabela é possível observar que há um claro aumento no tempo de cálculo das trajetórias, aliado obviamente aos melhores resultados obtidos. O melhor cenário

	Movimento da posição home para o ponto de corte				Nsucessos
	Tmax (s)	Tmin (s)	Tmed (s)	Nsucessos	total
BKPIECE	7,993	2,551	4,399	18/20	18/20
BiEST	3,97	0,867	2,354	19/20	19/20
EST	47,653	1,778	16,79	20/20	19/20
KPIECE	57,422	2,497	17,801	20/20	20/20
LBKPIECE	7,031	2,681	4,426	19/20	18/20
LazyPRMstar	2	2	2	19/20	17/20

Tabela 5.7: Tabela com os melhores resultados do 1º movimento do 2º teste do 1º cenário

	Movimento do ponto de corte para a posição home				Nsucessos
	Tmax (s)	Tmin (s)	Tmed (s)	Nsucessos	total
BKPIECE	5,708	1,439	3,191	18/18	18/20
BiEST	4,312	0,755	2,062	19/19	19/20
EST	3,086	0,865	1,8	19/20	19/20
KPIECE	5,426	0,83	2,072	20/20	20/20
LBKPIECE	7	1,649	3,917	17/18	18/20
LazyPRMstar	2	2	2	17/19	17/20

Tabela 5.8: Tabela com os melhores resultados do 2º movimento do 2º teste do 1º cenário

possível para o primeiro movimento comporta um tempo médio de 2s e 19 em 20 tentativas planeadas com sucesso, ao passo que no segundo movimento o tempo despendido é de 2,072s para um planeamento de movimentos perfeito, o que em ambos os casos significa um aumento de mais de 8 vezes o tempo computacional despendido.

Embora no primeiro movimento haja dois algoritmos que apresentaram uma assertividade perfeita, estes foram excluídos do grupo de testes a integrar o quinto teste. Considerou-se como critério para a sua exclusão o facto de o *trade-off* entre assertividade total e tempos 8 vezes superiores aos despendidos para o melhor cenário deste teste não justificar a sua consideração, uma vez que são bastante lentos e diminuiriam o rendimento do robô. Assim, relativamente ao primeiro movimento, foram escolhidos os algoritmos *BiEST*, *LBKPIECE* e *LazyPRMstar* para integrarem o quinto teste, uma vez que se considerou que podem efetivamente ser soluções.

Relativamente ao segundo movimento, os resultados são significativamente melhores que os do primeiro, sendo que, em média, todos os planeadores tomam no máximo 4s para o cálculo de trajetórias. Ainda assim, os algoritmos *BiEST* e *KPIECE* destacam-se dos demais, pelo que foram selecionados para integrar o quinto teste. Nota aqui para o facto do algoritmo *BiEST* ter sido selecionado em ambos os movimentos, o que é um bom indicador da sua performance nesta configuração.

De modo geral todos os algoritmos melhoraram os seus desempenhos, tanto em termos temporais como de assertividade, fruto da alteração do valor do parâmetro *longest_valid_segment_fraction*. Uma possível justificação para isto é o facto de, nesta configuração, a distância entre os *waypoints* das trajetórias estar limitada a valores da gama das grossuras dos troncos, o que indicia trajetórias

mais suaves nas zonas mais perto dos pontos de corte, que por sua vez justificam as colisões de raspão se verificaram, bem como a diminuição do número de colisões no geral. Assim, ainda que isto implique um maior esforço computacional, começa-se a desenvolver a ideia de que esta gama de valores temporais ajusta-se melhor que a obtida no primeiro teste para aplicações deste género, que implicam um alto valor de assertividade.

Com vista à melhoria dos resultados, foi novamente diminuído o valor do parâmetro em questão para a realização do terceiro teste, cujos resultados se apresentam de seguida.

5.3.3 Teste com o valor de 5*mm* no parâmetro *lvsf*

No terceiro teste do primeiro cenário todos os algoritmos são novamente testados com o valor de 5*mm* no parâmetro *longest_valid_segment_fraction*. Para os planeadores não limitados em tempo, o valor máximo definido para o planeamento manteve-se em 1 minuto, enquanto que os planeadores que estão limitados temporalmente dispõem agora de 3 segundos. Os resultados podem ser consultados nas linhas 68 a 87 das tabelas anexadas a esta dissertação, em baixo da célula *TEST 3*, preenchida a azul claro. Os dados da tabela 5.9 apresentam os mais relevantes resultados que são de seguida analisados.

Nº de algoritmos que planearam trajetórias que não entram em colisão com a vinha no máximo 4 vezes	10
Nº de algoritmos que conseguiram planear trajetórias livres de colisões 14 ou mais vezes	10
Nº de algoritmos cujas trajetórias calculadas colidem com a vinha no máximo 3 vezes	6
Nº de algoritmos que não encontraram solução pelo menos 10 vezes	10
Nº de algoritmos que nunca encontraram solução	6
Nº de algoritmos cujas trajetórias calculadas colidiam com a árvore 10 ou mais vezes	0
Nº de algoritmos que calcularam trajetórias não naturais	0
Nº total de colisões com a vinha	28
Nº de colisões no primeiro movimento	19
Nº de colisões no segundo movimento	9
Nº total de soluções não encontradas	200
Nº de solução não encontradas no primeiro movimento	185
Nº de solução não encontradas no segundo movimento	15
Nº de algoritmos onde se verificou o encerramento súbito do nó do *MoveIt!*	1

Tabela 5.9: Sumário de resultados do terceiro teste

À semelhança dos resultados do segundo teste, também aqui se verificam resultados bastante positivos em relação ao primeiro. Ainda assim, em comparação com o segundo teste é possível evidenciar um aumento dos valores temporais registados, assim como do número de vezes que não

foi encontrada solução. Por outro lado, o número de colisões também diminuiu, fruto da maior precisão das trajetórias planeadas.

Dos 23 algoritmos, 10 calcularam trajetórias que não entram em colisão com a vinha pelo menos 14 vezes, sendo que os algoritmos *BKPIECE*, *BiEST*, *EST*, *FMT*, *LBKPIECE* e *SBL* falharam no máximo 3 vezes, pelo que foram selecionados, com esta configuração do parâmetro *longest_valid_segment_fraction*, para o grupo de planeadores que poderá estar incluído no quinto reste. Nenhum é limitado temporalmente, e também não se registaram resultados perfeitos de assertividade, contrariamente ao que se sucedeu no segundo teste. Os piores resultados são os dos algoritmos *BFMT*, *BiTRRT*, *LBTRRT*, *PRM*, *PRMstar*, *RRT*, *RRTstar SPARS*, *SPARStwo* e *TRRT*, que conseguiram no máximo planear corretamente 4 vezes.

Os algoritmos *BiTRRT*, *LBTRRT*, *LazyPRM*, *PRM*, *PRMstar*, *RRT*, *RRTstar*, *SPARS*, *SPARStwo* e *TRRT* que não tinham encontraram solução pelo menos 14 vezes no segundo teste, baixaram este valor para 10. Ainda nesta situação, aos algoritmos *LBTRRT*, *RRT*, *RRTstar*, *SPARS* e *SPARStwo*, que não encontraram nenhuma solução no primeiro teste, juntou-se o algoritmo *BiTRRT*. No cômputo geral, e tal como afirmado anteriormente, o número total destes casos aumentou para 200, sendo verificados 185 no primeiro movimento e 15 no segundo. Assim, ainda que nos 10 algoritmos citados, o número mínimo destes casos tenha baixado para 10, verificou-se uma distribuição destas ocorrências por um maior número de algoritmos.

Tal como aconteceu do primeiro para o segundo teste, em relação a este último o número de impactos com a vinha, quer no primeiro quer no segundo movimento diminuiu, sendo os novos números agora de 19 para o primeiro movimento e 9 no segundo. Verificou-se novamente a ausência total de trajetórias atípicas e os impactos passaram todos a ser de raspão. Estes resultados provam mais uma vez que as trajetórias passam a ser mais suaves e detalhadas quanto menor for o valor do parâmetro configurado.

As situações de encerramento súbito do nó voltaram a verificar-se apenas no algoritmo *BFMT*.

5 algoritmos melhoraram o seu desempenho em termos de assertividade, segundo os valores descritos na tabela 5.10. Os restantes pioraram os seus desempenhos, exceto no caso do algoritmo *ProjEST* e dos que novamente não foram capazes de planear corretamente uma única vez, onde o resultado se manteve igual.

	2º teste	3º teste
BKPIECE	18	19
FMT	14	17
LBKPIECE	18	19
LazyPRM	5	10
SBL	16	19

Tabela 5.10: Algoritmos e valores de desempenho que melhoraram do 2º para o 3º teste

Ao contrário da situação registada no teste passado, estes dados revelam que poucos foram os casos em que a assertividade melhorou. Isto coincide com o que já foi dito anteriormente de que o número de situações em que o planeamento de trajetórias falhou, aumentou. Isto deve-se ao

facto do tempo necessário para este processo ter aumentado, tal como demonstram os valores das tabelas 5.11 e 5.12, para o primeiro e segundo movimentos, respetivamente, e referentes ao grupo de planeadores que poderá integrar o 5º teste.

	Movimento da posição home para o ponto de corte				Nsucessos
	Tmax (s)	Tmin (s)	Tmed (s)	Nsucessos	total
BKPIECE	11,567	4,135	7,966	19/20	19/20
BiEST	5,772	1,554	3,43	17/20	17/20
EST	53,748	3,106	24,83	18/20	18/20
FMT	59,117	45,793	50,913	18/20	17/20
LBKPIECE	15,06	3,938	8,561	19/20	19/20
SBL	5,287	2,691	3,826	19/20	19/20

Tabela 5.11: Tabela com os melhores resultados do 1º movimento do 3º teste do 1º cenário

	Movimento do ponto de corte para a posição home				Nsucessos
	Tmax (s)	Tmin (s)	Tmed (s)	Nsucessos	total
BKPIECE	13,931	3,018	7,547	19/18	19/20
BiEST	5,46	0,926	3,121	17/17	17/20
EST	8,471	0,783	3,715	18/18	18/20
FMT	47,4	36,641	41,443	17/18	17/20
LBKPIECE	12,751	3,129	8,106	19/19	19/20
SBL	5,984	1,808	3,713	19/19	19/20

Tabela 5.12: Tabela com os melhores resultados do 2º movimento do 3º teste do 1º cenário

Analisando os resultados da tabela é possível observar que há um novo aumento no tempo de cálculo das trajetórias. No entanto, os resultados não melhoraram relativamente ao teste anterior pois o melhor cenário possível para o primeiro movimento é agora de um de 3,826s e 19 em 20 tentativas planeadas com sucesso, ao passo que no segundo movimento o tempo despendido é de 3,121s para 17 em 17 tentativas bem sucedidas, o que em ambos os casos significa um aumento de mais de 1s. Enquanto que no anterior teste não haviam médias superiores a 18s, neste há dois casos onde a média é superior a 24s, havendo até um caso de quase 51s, ambos no primeiro movimento.

Nesse primeiro movimento não foi registado nenhum caso de assertividade perfeita, sendo que dos dois algoritmos que a registaram no teste anterior, um aumentou em 8s o seu tempo médio computacional, enquanto que o outro nem se encontra nesta tabela pois apenas foi capaz de efetuar corretamente o teste 14 vezes. Este é um indicador de que os valores do parâmetro alterado nem sempre aumentam a assertividade dos planeadores. Uma vez que nenhum dos resultados apresenta melhores valores temporais e de assertividade em relação aos do segundo teste, nenhuma destas configurações de planeadores foi selecionada para integrar o quinto teste.

Relativamente ao segundo movimento, foi registado mais um caso de assertividade perfeita em relação ao mesmo movimento do segundo teste. Ainda assim, e de forma semelhante ao primeiro

movimento, nenhum dos resultados apresenta melhores valores temporais e de assertividade superiores aos do segundo teste, pelo que nenhuma destas configurações de planeadores foi selecionada para integrar o quinto teste.

Relativamente ao último teste, o desempenho dos algoritmos neste teste piorou tanto em termos temporais como de assertividade. Ainda que fosse de esperar uma assertividade superior, estando os algoritmos limitados a tempos computacionais de 1m, é natural que necessitem de mais tempo para calcular as trajetórias, o que leva a um aumento dos casos onde a solução não foi encontrada. No entanto, este limite não será mais aumentado, uma vez que o seu principal propósito é precisamente estudar se os algoritmos necessitam desta gama de valores nas suas operações, o que se tornaria inviável na realidade. Ainda assim, e como seria de esperar, as trajetórias tornaram-se ainda mais suaves, não tendo sido registados impactos na vinha que a fizessem cair.

Com vista à melhoria dos resultados, foi novamente diminuído o valor do parâmetro em questão. São de seguida apresentados os resultados do quarto teste.

5.3.4 Teste com o valor de 1*mm* no parâmetro *lvsf*

No quarto teste do primeiro cenário todos os algoritmos são novamente testados com o valor de 1*mm* no parâmetro *longest_valid_segment_fraction*. Para os planeadores não limitados em tempo, o valor máximo definido para o planeamento manteve-se em 1 minuto, enquanto que os planeadores que estão limitados temporalmente dispõem agora de 4 segundos. Os resultados podem ser consultados nas linhas 98 a 117 das tabelas, em baixo da célula *TEST 4*, preenchida a azul claro. Os dados da tabela 5.13 apresentam os mais relevantes resultados que são de seguida analisados.

Nº de algoritmos cujas trajetórias calculadas colidem com a vinha no máximo 3 vezes	6
Nº de algoritmos que nunca encontraram solução	12
Nº de algoritmos cujas trajetórias calculadas colidiam com a árvore 10 ou mais vezes	0
Nº de algoritmos que calcularam trajetórias não naturais	0
Nº total de colisões com a vinha	7
Nº de colisões no primeiro movimento	5
Nº de colisões no segundo movimento	2
Nº total de soluções não encontradas	295
Nº de solução não encontradas no primeiro movimento	266
Nº de solução não encontradas no segundo movimento	29
Nº de algoritmos onde se verificou o encerramento súbito do nó do MoveIt!	0

Tabela 5.13: Sumário de resultados do quarto teste

Em comparação com os dois últimos testes, é possível evidenciar um aumento dos valores temporais registados, assim como do número de vezes que não foi encontrada solução. Por outro

lado, o número de colisões também diminuiu novamente, sendo escassos os casos registados.

Dos 23 algoritmos, apenas se destacam o *BKPIECE*, o *BiEST*, o *LBKPIECE*, o *RRTConnect*, o *SBL* e o *STRIDE*, que falharam no máximo 3 vezes, pelo que foram selecionados, com esta configuração do parâmetro *longest_valid_segment_fraction*, para o grupo de planeadores que poderá estar incluído no quinto reste. Os melhor resultados a seguir a estes são os dos algoritmos *PDST* e *LazyPRMstar*, com o respetivo registo de 12 e 9 intentos bem sucedidos num total de 20. Novamente nenhum é limitado temporalmente, sendo que neste teste se registaram 4 que efetuarem o teste com assertividade perfeita, melhorando o anterior melhor resultado que foi atingido unicamente pelo algoritmo *KPIECE*, no segundo teste. Os restante planeadores que não foram citados não foram capazes de planear corretamente mais de 4 vezes.

Os algoritmos *BMFT*, *BiTRRT*, *EST*, *FMT*, *KPIECE LBTRRT*, *PRMstar*, *ProjEST*, *RRT*, *RRTstar*, *SPARS* e *SPARStwo* não encontraram qualquer solução durante o teste. No cômputo geral, e tal como afirmado acima, o número total destes casos aumentou para 295, sendo verificados 266 no primeiro movimento e 29 no segundo, o que evidencia a capacidade da maior parte dos planeadores de não conseguir computar as trajetórias no tempo que têm disponível.

Tal como aconteceu do primeiro para o segundo teste, em relação a este último o número de impactos com a vinha, quer no primeiro quer no segundo movimento diminuiu, sendo os novos números agora de 5 para o primeiro movimento e 2 no segundo. Verificou-se novamente a ausência total de trajetórias atípicas e os poucos impactos foram todos de raspão. Estes resultados provam mais uma vez que as trajetórias passam a ser mais suaves e detalhadas quanto menor for o valor do parâmetro configurado.

As situações de encerramento súbito do nó deixaram de se verificar no algoritmo *BFMT*, pelo que não foi verificada qualquer ocorrência neste teste. Por sua vez, este algoritmo não encontrou nenhuma solução.

Apenas os algoritmos *BiEST*, *LBKPIECE*, *RRTConnect*, *SBL* e *STRIDE* melhoraram as suas assertividades, sendo que todos pertencem ao grupo candidato ao quinto teste e os 4 últimos registaram valores perfeitos. Os restantes pioraram os seus desempenhos, à exceção dos que novamente não foram capazes de planear corretamente uma única vez, onde o resultado se manteve nulo. São de seguida apresentados nas tabelas 5.14 e 5.15, para o primeiro e segundo movimentos, respetivamente, os resultados referentes ao grupo de planeadores que poderá integrar o 5º teste.

	Movimento da posição home para o ponto de corte				Nsucessos
	Tmax (s)	Tmin (s)	Tmed (s)	Nsucessos	total
BKPIECE	54,631	22,493	39,791	19/20	18/20
BiEST	48,866	5,32	22,398	19/20	19/20
LBKPIECE	58,299	19,794	39,707	20/20	20/20
RRTConnect	16,806	4,874	11,573	20/20	20/20
SBL	25,456	12,957	18,372	20/20	20/20
STRIDE	23,459	9,153	18,057	20/20	20/20

Tabela 5.14: Tabela com os melhores resultados do 1º movimento do 4º teste do 1º cenário

	Movimento do ponto de corte para a posição home				Nsucessos total
	Tmax (s)	Tmin (s)	Tmed (s)	Nsucessos	
BKPIECE	56,34	10,584	33,258	17/18	18/20
BiEST	38,373	6,34	20,981	19/19	19/20
LBKPIECE	58,849	11,199	42,786	20/20	20/20
RRTConnect	19,247	6,484	11,258	20/20	20/20
SBL	23,803	8,585	17,383	20/20	20/20
STRIDE	23,657	13,82	18,417	20/20	20/20

Tabela 5.15: Tabela com os melhores resultados do 2º movimento do 4º teste do 1º cenário

Analisando os resultados da tabela é possível observar que há um novo aumento no tempo de cálculo das trajetórias. No entanto, os resultados voltaram a não melhorar relativamente ao teste anterior pois o melhor cenário possível para o primeiro movimento é agora de um de 11,573s ao passo que no segundo movimento o tempo despendido é de 11,258s, ambos com assertividades perfeitas. No entanto, uma vez que nenhum dos resultados apresenta melhores valores temporais e de assertividade conjunta em relação aos do segundo teste, nenhuma destas configurações de planeadores foi selecionada para integrar o quinto teste.

Relativamente ao último teste, o desempenho dos algoritmos neste teste piorou em termos temporais, tendo apenas melhorado na assertividade. Os resultados estão assim de acordo com a premissa de que um maior pormenor nas trajetórias requer um valor mais pequeno do parâmetro *longest_valid_segment_fraction*. No entanto, os tempos computacionais tornam-se maiores e pouco viáveis, o que leva a descartar os algoritmos com esta configuração.

Os algoritmos escolhidos para integrarem o quinto teste são o *BiEST*, *LBKPIECE* e *LazyPRMstar* para o primeiro movimento e o *BiEST* e *KPIECE* para o segundo movimento, configurados com o valor do parâmetro usado no segundo teste. Os seus resultados são apresentados de seguida

5.3.5 Teste com 5 pontos na vinha

No quinto teste do primeiro cenário de testes a plataforma AGROB V16 move-se ao da vinha procurando que o seu manipulador atinja os 5 pontos de corte por ela distribuídos. Neste teste é efetuado o *benchmarking* dos algoritmos *BiEST*, *LBKPIECE* e *LazyPRMstar* para o primeiro movimento e *BiEST* e *KPIECE* para o segundo movimento, todos configurados com o valor de 1*mm* no parâmetro *longest_valid_segment_fraction*. O objetivo final deste teste é chegar às melhores configurações a ser testadas com o modelo de vinha complexo.

Para os planeadores não limitados em tempo o valor máximo definido para o planeamento diminuiu para 10s, enquanto que o algoritmo *LazyPRMstar*, o único limitado temporalmente, dispõem apenas de 2 segundos. Os algoritmos escolhidos para os dois movimentos foram combinados gerando um total de 6 diferentes configurações. Os resultados podem ser consultados nas linhas 193 a 202 das tabelas, em baixo da célula *TEST 5*, preenchida a azul claro. De notar que,

contrariamente aos restantes testes que analisavam individualmente cada movimento, neste caso um teste completado com sucesso implica que sejam efetuados 5 movimentos da posição *home* do manipulador para diversos pontos de corte e outros 5 movimentos das posições de corte para a posição *home*.

A primeira impressão a retirar dos resultados é que há bastantes colisões com a vinha e apenas uma situação em que o planeamento de trajetórias não encontrou solução. Os resultados demonstram desta forma que nem todas as configurações apresentam resultados ótimos, como eventualmente seria de esperar, e que pontos em diferentes posições, com diferentes orientações e que possam estar distantes do manipulador, alteram a ideia inicial de quais seriam os melhores planeadores. Ainda que isto pudesse remeter para o teste de outros algoritmos neste ambiente, o peso que as questões temporais têm remete para que o mesmo não se suceda. Os resultados dos testes nas diferentes configurações apresentam-se na tabela 5.16.

	Tmax (s)	Tmin (s)	Tmed (s)	TmedP (s)	Nsucessos total
BiEST + BiEST	32,716	20,723	26,416	5,283	5/10
BiEST + KPIECE	29,111	21,62	26,192	5,238	7/10
LBKPIECE + BiEST	45,471	32,623	38,275	7,655	5/10
LBKPIECE + KPIECE	43,399	38,197	39,366	7,873	5/10
LazyPRMstar + BiEST	33,008	18,42	23,474	4,694	10/10
LazyPRMstar + KPIECE	27,828	18,757	21,718	4,344	9/10

Tabela 5.16: Tabela com os melhores resultados do 5º teste do 1º cenário

De notar que o tempo médio por ponto é igual ao tempo médio total dividido por 5 e que o símbolo + separa o nome dos algoritmos usados nos primeiro e segundo movimentos, respetivamente.

A configuração que consiste no uso do algoritmo *BiEST* nos dois movimentos e a que usa o algoritmo *LBKPIECE* no primeiro obtiveram uma taxa de assertividade de 50%, a mais baixa de entre todos. Além disso, estes dois últimos casos obtiveram os piores resultados temporais, pelo que foram automaticamente excluídos do teste com a vinha complexa. A configuração que usou o algoritmo *BiEST* no primeiro movimento e *KPIECE* no segundo teve também uma taxa de assertividade de 70% pelo que também foi excluída.

Os melhores resultados obtidos, tanto em termos de assertividade como em termos temporais, foram registados pelas duas configurações que usam o algoritmo *LazyPRMstar* no primeiro movimento. No primeiro caso a assertividade foi perfeita, enquanto que no segundo apenas houve uma colisão no segundo movimento, pelo que o algoritmo *LazyPRMstar*, com o valor de $1cm$ no parâmetro *longest_valid_segment_fraction* é claramente a melhor opção para o primeiro movimento. Em relação ao segundo movimento, o algoritmo *BiEST* foi superior ao *KPIECE* em termos de assertividade, ainda que seja ligeiramente mais lento. Por estas razões, estas duas configurações foram escolhidas para ser testadas no segundo movimento com a vinha complexa.

Além das considerações relativas aos algoritmos analisados, foi possível analisar o comportamento do *MoveIt!* nas situação em que o manipulador não se encontra próximo do ponto de corte (isto é, a escassos *cm*), dado que se tem que deslocar ao longo da vinha para aceder a outros.

Numa situação em que um ponto de corte se encontre à frente do manipulador e dentro do seu volume de atuação, seria de esperar que o *MoveIt!* conseguisse planear para ele, independentemente da sua orientação, e desde que não houvesse objetos intransponíveis que impossibilitassem qualquer planeamento. A figura 5.9 demonstra uma situação deste tipo.

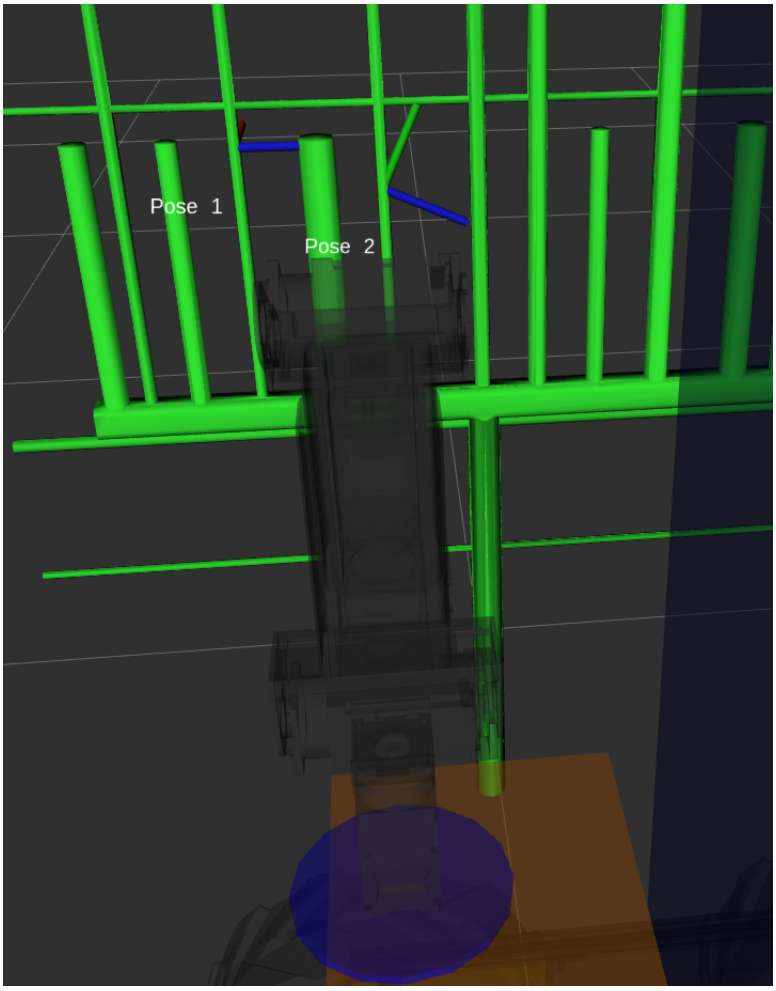


Figura 5.9: Situação onde o manipulador se encontra em frente ao ponto e não consegue computar nenhuma trajetória

Ainda que o esperado fosse o *MoveIt!* conseguir planear na situação da figura, o mesmo apenas foi conseguido quando o AGROB V16 já se encontrava a cerca de 20*cm* do ponto, no sentido positivo do eixo dos *xx*. A posição em que a plataforma se encontra quando o manipulador atinge o ponto é representada na figura 5.10.

A particularidade deste ponto é o facto das lâminas da tesoura de poda se encontrarem a apontar para o chão. Ainda no ponto 5 deste mesmo teste ocorreu a mesma situação, sendo que neste caso o AGROB V16 se encontrava antes do ponto cerca de 30*cm* na direção do eixo dos *xx*.

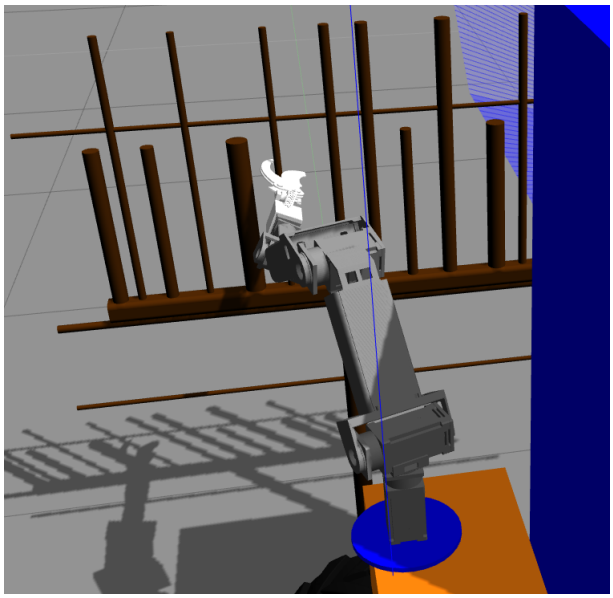


Figura 5.10: Posição do AGROB V16 quando o *MoveIt!* conseguiu planear para o 2º ponto no quinto teste do primeiro cenário

A figura 5.11 apresenta a posição da plataforma no momento em que o último ponto deste teste foi alcançado.

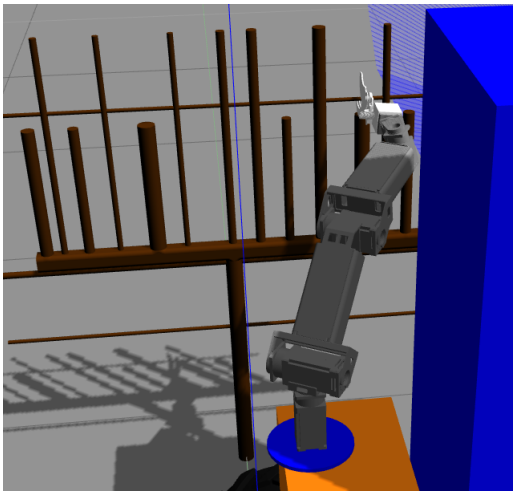


Figura 5.11: Posição do AGROB V16 quando o *MoveIt!* conseguiu planear para o 5º ponto no quinto teste do primeiro cenário

Estes 2 resultados remetem para a premissa de que o *MoveIt!* não consegue planear para um ponto apenas porque este se encontra ao alcance do manipulador, mesmo que esta pareça perfeitamente viável. Outra particularidade destes casos é o facto de em todos os algoritmos se terem verificado comportamentos semelhantes, não havendo nenhum caso onde o *MoveIt!* tivesse conseguido calcular as trajetórias com o AGROB V16 em posições diferentes das verificadas nas imagens.

É de seguida apresentado o segundo cenário, o teste nele realizado e a sua análise de resultados.

5.4 Modelo AGROB V16 e vinha complexa

O segundo cenário de testes é constituído pelo modelo do AGROB V16 e da vinha complexa, ambos simulados em *Gazebo*. A plataforma AGROB V16 começa todos os testes na posição $(x,y,z) = (0, 0, 0)$ e a vinha encontra-se a $80cm$ da plataforma, na direção positiva do eixo dos yy. A figura 5.12 apresenta uma imagem deste cenário no *Gazebo*.

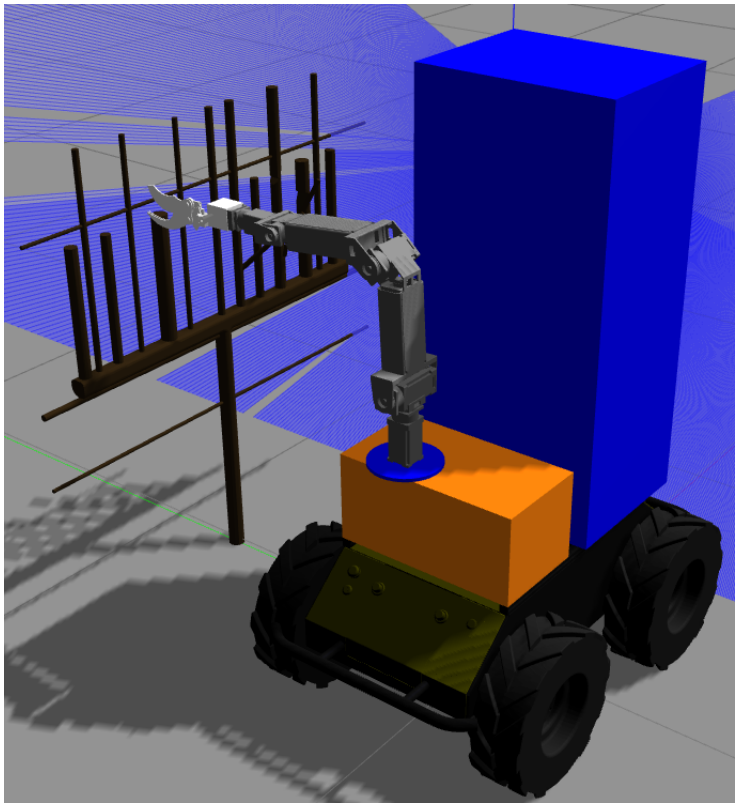


Figura 5.12: Segundo cenário de testes

De notar que, ainda que não sejam bem visíveis as claras diferenças entre os 2 modelos de vinha, as cores de ambas são diferentes.

Neste cenário apenas foi realizado um teste que consiste em posicionar o *end effector* do manipulador em 3 diferentes pontos dispostos na zona da vinha com troncos dispostos em direções aleatórias. Sempre que um ponto for atingido, o manipulador deve voltar à sua posição inicial antes de iniciar nova tentativa de planeamento. Um teste bem sucedido é aquele em que as 6 trajetórias são planeadas de forma a não chocarem com a vinha. Na figura 5.13 apresentam-se as posições finais da tesoura nos 3 pontos de corte. De notar que neste caso os pontos não são apresentados como na figura 5.7, uma vez que são muito próximos um dos outros e pouco percetíveis no *RViz*.

Este último teste destina-se a avaliar as duas configurações que obtiveram melhores resultados no 5^{o} teste do 1^{o} cenário, pelo que no planeamento de trajetórias do movimento de ida para os

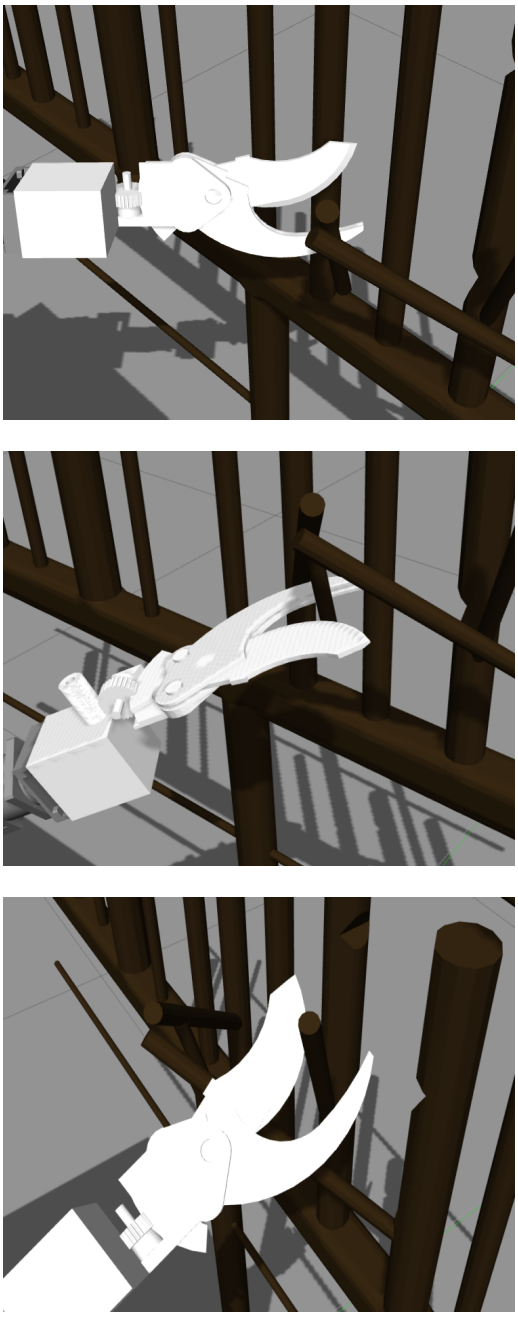


Figura 5.13: 3 pontos de corte no modelo de vinha complexa

pontos de corte foi apenas utilizado o algoritmo *LazyPRMstar*, ao passo que no segundo movimento são usados os algoritmos *BiEST* e *KPIECE*, ambos configurados com um valor de $1cm$ no parâmetro *longest_valid_segment_fraction*

Foram efetuados 10 testes para cada combinação, o que totaliza 30 trajetórias calculadas. O AGROB V16 desloca-se ao longo da vinha cada vez que um plano para o primeiro movimento não consiga ser calculado. Uma vez que foi utilizado o algoritmo *LazyPRMstar* para o seu planeamento, e que este está limitado a 2s para planear as trajetórias, a plataforma desloca-se de 2 em 2s de cada vez que o planeamento falha. Para os restantes algoritmos um tempo máximo de 15s

foi estipulado como limite para o cálculo das trajetórias.

Os resultados deste teste são apresentados na tabela 5.17.

	Tmax (s)	Tmin (s)	Tmed (s)	TmedP (s)	Nsucessos total
LazyPRMstar + BiEST	27,484	12,539	19,368	6,456	9/10
LazyPRMstar + KPIECE	7,356	7,016	7,173	2,391	3/10

Tabela 5.17: Tabela com os melhores resultados do teste no 2º cenário

De notar que o tempo médio por ponto é igual ao tempo médio total dividido por 3 e que o símbolo + separa o nome dos algoritmos usados nos primeiro e segundo movimentos, respetivamente.

A primeira configuração testada teve uma taxa de sucesso de 90%, sendo que a segunda baixou o seu valor para 30%, tendo ambos os resultados piorado em relação aos do quinto teste do primeiro cenário. Ainda em relação a este teste, os tempos obtidos foram piores no primeiro caso, e ainda, que tenham sido melhores no segundo, esta configuração foi excluída devida à baixa assertividade.

Em termos de colisões, e há semelhança dos anteriores resultados em que o algoritmo *LazyPRMstar* foi escolhido para o primeiro movimento, a única ocorrência verificou-se no segundo movimento. Ainda assim, o algoritmo *BiEST* continua claramente a apresentar os melhores resultados, pelo que a combinação dos dois algoritmos se apresenta aqui como sendo a melhor para este tipo de aplicação.

Tal como efetuado no anteriores teste, também neste foi possível analisar o comportamento do *MoveIt!* nas situações em que o manipulador não se encontra próximo do ponto de corte. Todos os pontos de corte foram alcançados em posições da plataforma que não colocavam o manipulador em frente ao ponto, como mostra a figura 5.14.

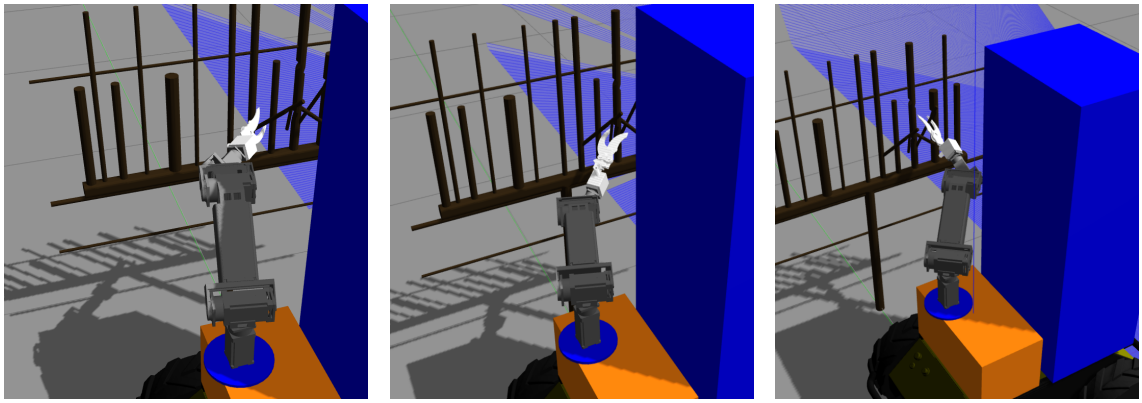


Figura 5.14: Posição do AGROB V16 no momento em que foram calculadas as trajetórias para os 3 pontos de corte no modelo de vinha complexa

Através destes resultados e dos do último teste é possível perceber que a posição da plataforma tem de facto influência na tarefa de planeamento de movimentos, sendo que o *MoveIt!* consegue

planear para um ponto (ainda que o plano inclua um trajeto que colida com a vinha) desde que o AGROB V16 se encontre no interior de uma região que varia de ponto para ponto e que, aparentemente, não pode ser estimada. Ainda que isto seja um aspeto negativo, é possível contornar esta situação movendo a plataforma entre regiões perto do ponto. Por outro lado, isto também indica que o planeamento seja possível com terrenos agrestes típicos de cultura de vinha pois, desde que seja possível transmitir ao *MoveIt!* que o AGROB V16 se encontra inclinado e desde que este se encontre dentro desta região de planeamento (ou seja, que o ponto esteja ao alcance do manipulador), é possível planear para esse ponto.

Como forma de sumarizar toda a análise efetuada nos dois cenários de testes, segue-se um subcapítulo que resume os resultados obtidos.

5.5 Sumário de resultados

Neste capítulo foi realizado o *bechmarking* dos 23 algoritmos de planeamento de trajetórias da *OMPL* implementados no *MoveIt!*. O ambiente de teste consistiu numa simulação em *Gazebo* onde um modelo do *AGROB V16* se encontra na origem e um modelo de vinha se encontra a 80*cm* deste, no sentido positivo do eixo dos *yy*.

Foram efetuados testes com dois modelos de vinha, pelo que à simulação com o modelo de vinha simples foi dado o nome de primeiro cenário e à simulação com o modelo de vinha complexo foi dado o nome de segundo cenário. Uma vez que os existem 36 diferentes parâmetros que podem ser alterados entre todas os algoritmos do *MoveIt!*, apenas foi estudado a influência do parâmetro *longest_valid_segment_fraction* nas trajetórias calculadas, considerado como o mais importante para o presente estudo. O objetivo principal dos testes foi, através da alteração deste parâmetro, encontrar a combinação de planeadores que melhores resultados apresenta para os movimentos da posição *home* do manipulador para os pontos de corte, e vice-versa.

Foram realizados 5 testes no primeiro cenário, sendo que nos primeiros 4 existe apenas um ponto de corte e que é alcançável pelo manipulador desde a posição de origem do AGROB V16. Foram efetuadas 20 tentativas para realizar os 2 movimentos neste ponto, com o objetivo de estudar a influência do parâmetro *longest_valid_segment_fraction* em todos os algoritmos. Este parâmetro impõe a distância máxima entre *waypoints* das trajetórias geradas e os valores que lhe foram atribuídos ao longo dos 4 primeiros testes foram de 5 (valor original), 1, 0.5 e 0.1*cm*, respetivamente. Para os algoritmos que tomam sempre o tempo máximo disponível para planear trajetórias foram atribuídos os valores de 1, 2, 3 e 4s, respetivamente, em cada teste, sendo que os restantes dispuseram sempre de, no máximo, 1m para a mesma operação.

No último teste deste cenário, os melhores planeadores para cada movimento nos 4 primeiros testes foram testados em 5 pontos dispostos ao longo da vinha, num total de 10 tentativas. Neste caso, a posição dos pontos exige a movimentação do AGROB V16 ao longo da vinha, de forma a

que o manipulador na sua base os consiga alcançar a todos. São admitidas neste teste as configurações que tenham sucedido em planear pelo menos 17 em 20 tentativas e que apresentem os mais baixos valores médios temporais de planeamento.

No segundo cenário foi apenas realizado um teste com as configurações que melhores resultados obtiveram no 5º teste do primeiro cenário. Neste teste são novamente realizadas 10 tentativas de atingir 3 pontos disposto na zona da vinha onde os troncos se encontram em posições aleatórias.

Nos primeiros 4 testes foi possível determinar que os algoritmos *BFMT*, *BiTRRT*, *LBTRRT*, *LazyPRM*, *PDST*, *PRM*, *PRMstar*, *ProjEST*, *RRT*, *RRTstar*, *SPARS*, *SPARStwo* e *TRRT* nunca foram capazes de planear corretamente mais de 16 vezes em 20, pelo que foram excluídos das configurações a ser testadas no último teste deste cenário.

O algoritmo *BKPIECE*, que apenas teve sucesso em 7 tentativas no primeiro teste, obteve 18, 19 e 18 tentativas bem sucedidas nos três seguintes testes. Ao longo dos 4 testes verificou-se ainda o aumento do tempo médio necessário para calcular os 2 movimentos, sendo que, como se verificaram valores inferiores aos deste algoritmo para outros planeadores, este não foi escolhido para integrar o quinto teste.

Um comportamento semelhante teve o algoritmo *BiEST*, onde o número de sucessos foi de 5, 19, 17 e 19, respetivamente, para os primeiros 4 testes. Novamente os tempos médios em ambos os movimentos foram aumentando ao longo do testes, sendo que no 2º foi obtido o 2º valor mais baixo para o primeiro movimento (2.354s) e o valor mais baixo no segundo movimento (2.032s), pelo que este planeador, com este valor do parâmetro, foi selecionado para ambos os movimentos o quinto teste.

O algoritmo *EST* obteve 3, 19, 18 e 0 sucessos, respetivamente, nos primeiros 4 testes. À semelhança dos algoritmos já referidos, os valores das médias temporais em ambos os movimentos tenderam a aumentar com a diminuição do valor do parâmetro, de tal forma que nenhuma trajetória foi computada no quarto teste, devido à falta de mais tempo disponível. Apesar dos bons valores nas taxas de sucesso, os valores das médias temporais impossibilitaram este planeadores de integrar o próximo teste.

O algoritmo *FMT* foi o primeiro onde se verificou o incremento de valores de sucesso nas tentativas, até um que fosse viável para incluir este algoritmo no quinto teste. Estes valores foram de 8, 14 e 17, sendo que no último teste não foi encontrada nenhuma solução pelos mesmo motivos que o algoritmo *EST*. Ainda assim, pelos mesmo motivos do anterior algoritmo, este também não integrou o seguinte teste.

O algoritmo *KPIECE* atingiu o seu melhor resultado ao computar com sucesso todas as tentativas no segundo teste, tendo o seu valor melhorado em 12 tentativas do primeiro teste para este. Uma vez que o tempo médio despendido no segundo movimento apresentou o 2º melhor valor entre todos (2.072s), este algoritmo, com esta configuração do parâmetro, foi escolhido para integrar os testes do segundo movimento do quinto teste. Nos seus dois últimos testes as tentativas bem sucedidas baixaram para 14 e 0.

Tal como o algoritmo *FMT*, o *LBKPIECE* melhorou os seus valores em todas as tentativas, sendo eles 8, 18, 19 e 20, respetivamente. No segundo teste este algoritmo apresentou o valor

de 4.413s em média (3º melhor valor) para computar o primeiro movimento, pelo que, com esta configuração do parâmetro de estudo, foi escolhido para integrar os testes deste movimento no quinto teste.

O algoritmo *LazyPRMstar* obteve o seu melhor resultado no segundo teste, tendo computado com sucesso 17 testes. Uma vez que é um planeador limitado temporalmente e como neste teste o limite temporal foi de 2s, esta configuração foi escolhida para integrar os testes do primeiro movimento do quinto teste como sendo a melhor entre todas. No primeiro, terceiro e quarto testes, as tentativas bem sucedidas foram 10, 15 e 9, respetivamente.

Os algoritmos *RRTConnect* foram os únicos casos *STRIDE* onde apenas no quarto teste foi atingido um valor de sucessos acima de 16, sendo esse valor de 20 em ambos. Ainda assim, nesta configuração ambos requerem tempos muito altos de planeamento, pelo que também foram excluídos do seguinte teste.

Por último, o algoritmo *SBL* obteve valores de 12, 16, 19 e 20 sucessos, respetivamente, ao longo dos seus 4 testes. Ainda assim, os seus altos valores temporais tornaram a sua implementação inviável.

Em relação aos 4 primeiros testes, no cômputo geral verificaram-se os seguintes comportamentos com a alteração dos valores do parâmetro *longest_valid_segment_fraction*:

- As situações onde foram computadas trajetórias não naturais (figura 5.8), apenas ocorreram para um valor de 5*cm* neste parâmetro;

- Para valores altos deste parâmetro (como por exemplo os 5*cm* predefinidos), as trajetórias geradas comportam menos *waypoints*, pelo que movimentos que implicam algum detalhe, como os da tesoura próximos dos troncos, não conseguem ser efetuados sem que esta entre em colisão com a vinha. Por esta razão, existem mais colisões com a vinha nestes casos e o tempo computacional requerido para o planeamento de trajetórias é mais baixo que nos restantes casos;

- Para valores mais baixos do parâmetro o detalhe das trajetórias aumenta, assim como o tempo requerido para as planear. Como consequência do maior detalhe, há uma diminuição do número de colisões, e o aumento do tempo de planeamento faz com que a maioria dos algoritmos seja incapaz de planear qualquer trajetória num minuto.

No quinto teste foram testados no primeiro movimento os algoritmos *BiEST*, *LBKPIECE* e *LazyPRMstar* combinados com os algoritmos *BiEST* e *KPIECE* no segundo movimento. Foram todos configurados com um valor de 1*cm* no parâmetro *longest_valid_segment_fraction*, visto que os seus melhores resultados foram registados no 2º teste. Há exceção do *LazyPRMstar*, a todos os restantes algoritmos foi dado 15s como tempo máximo para o planeamento.

Verificou-se neste teste que as únicas combinações de configurações que completaram o teste com pelo menos 9 sucessos em 10 foram aquelas que usaram o algoritmo *LazyPRMstar* no primeiro movimento, sendo que as restantes obtiveram valores inferiores a 7, pelo que foram excluídas do sexto e último teste. Nos dois melhores casos, a combinação que usou o algoritmo *BiEST* no segundo movimento teve uma taxa de assertividade perfeita, enquanto que a que usou o algoritmo *KPIECE* colidiu apenas uma vez com a vinha e no segundo movimento. Além do seu número de sucessos, estas combinações também apresentaram os melhores resultados temporais, tendo um tempo médio para o planeamento de 10 movimentos de 23.474s e 21.718s para os algoritmos *BiEST* e *KPIECE* no segundo movimento, respetivamente. Por estas razões, estas foram as duas combinações escolhidas para integrar o sexto teste.

Outro aspeto importante a salientar foi o tempo despendido para que o AGROB V16 se deslocasse rapidamente entre pontos, uma vez que foi definido que sempre que o planeamento do primeiro movimento falhasse, a plataforma deslocar-se-ia para a frente. Uma vez que, há exceção do algoritmo *LazyPRMstar*, todos os restantes tinham no máximo 15s para planear, os testes demoraram muito mais tempo a ser executados nestes casos, o que na prática inviabiliza a sua implementação. Assim, além do *LazyPRMstar* ter obtido os melhores resultados práticos em termos de assertividade e tempo, também permite que o AGROB V16 se desloque mais rapidamente entre pontos, aumentando a sua rentabilidade tanto energética como de pontos de corte alcançados com sucesso.

No sexto teste e único teste do segundo cenário foram testadas as duas combinações mencionadas em cima com o objetivo de atingirem pontos em posições e orientações que mais se assemelham aquelas que se espera encontrar numa vinha real.

Os resultados obtidos revelam que não houve taxas de sucesso perfeitas, sendo que no caso em que o algoritmo *KPIECE* foi usado no segundo movimento, a taxa foi de apenas 30%, o que descarta esta combinação como uma solução para problemática de estudo. A combinação que usou o algoritmo *BiEST* no segundo movimento colidiu com a vinha apenas uma vez e no segundo movimento, pelo que novamente se confirma que o planeador *LazyPRMstar* é a melhor opção para o primeiro movimento. Em termos temporais, o tempo médio total para os 6 movimentos foi de 19.368s, tendo subido em relação ao quinto teste. Isto remete para que pontos com orientações mais adversas compliquem o cálculo de trajetórias, podendo-se esperar, na prática, que os dois movimentos num ponto da vinha tomem cerca de 6,456s a serem computados.

Finalmente, a posição do AGROB V16, aquando do cálculo do primeiro movimento, foi analisada nos dois últimos testes. Foi possível perceber que a plataforma estar em frente ao ponto de corte não é condição suficiente para que o *MoveIt!* consiga planear para o ponto em questão, pelo que muitas vezes era necessário que esta deslocasse cerca de 10 a 20*cm* para a frente ou para trás de modo a tornar o planeamento possível. Ainda que não seja possível estimar a região que torna possível este planeamento, é válido afirmar que, desde que o ponto esteja ao alcance do manipulador, o *MoveIt!* conseguirá calcular um plano, ainda que seja necessário deslocar o AGROB V16 numa zona próxima do mesmo.

Resumidamente, os últimos dois testes permitiram concluir que:

- O algoritmo *LazyPRMstar* é o que melhor que resultados apresenta em relação ao primeiro movimento, tanto tem termos temporais como em termos de assertividade;
- A combinação que usa o algoritmo *LazyPRMstar* no primeiro movimento e *BiEST* no segundo, ambos configurados com o valor de $1cm$ no parâmetro *longest_valid_segment_fraction*, apresenta os melhores resultados numa aplicação de vinha;
- Desde que o ponto esteja ao alcance do manipulador, usando esta combinação o *MoveIt* encontrará sempre solução para o problema do planeamento de movimentos para os pontos de corte, ainda que haja a possibilidade (reduzida) da trajetória chocar de raspão com a vinha.

Após a análise geral dos testes em ambiente de simulação, é abordada a implementação do *MoveIt!* no sistema real.

5.6 Implementação na plataforma real

Uma vez realizados os testes em ambiente de simulação e conhecidos os resultados esperados, passou-se à implementação do sistema na plataforma real.

Ainda que não pudessem ser realizados testes devido ao facto da tesoura de poda real não se encontrar disponível, procedeu-se assim à implementação do *MoveIt!* no manipulador da plataforma AGROB V16.

Embora não exista nenhum *package* desenvolvido pelo fabricante para que o *Robotis Manipulator-H* interaja diretamente com o *MoveIt!*, a *Robotis* desenvolveu um *metapackage* que o faz para o seu mais recente manipulador, o *Open Manipulator Chain*[2], que se apresenta na figura 5.15[3].

Lançado em 2017[4], este manipulador é constituído por 5 componentes ligadas com 4 juntas, sendo o seu *end effector* uma garra. Tem em comum com o *Manipulator-H* o facto de ter todas as juntas rotativas e de estas conterem motores da marca *Dynamixel*[5], também desenvolvidos pela *Robotis*. Antes do lançamento deste manipulador, a *Robotis* não tinha lançado nenhum *package* que permitisse controlar estes servos através do *MoveIt!*, o que consequentemente permitiria controlar o manipulador. No entanto, os *packages* do *Open Manipulator Chain* permitem conectar diretamente o manipulador ao *MoveIt!* e operá-lo como descrito no subcapítulo 4.2.3.

Após consultar[6] a equipa de desenvolvimento da *Robotis* destacada para este manipulador, fui aconselhado por eles a alterar o código fonte do *metapackage* do *Open Manipulator Chain* de

[2] http://emanual.robotis.com/docs/en/platform/openmanipulator/
[3] https://github.com/ROBOTIS-GIT/open_manipulator
[4] https://vimeo.com/236147296
[5] http://www.robotis.us/dynamixel/
[6] https://github.com/ROBOTIS-GIT/dynamixel-workbench/issues/141#issuecomment-383752996

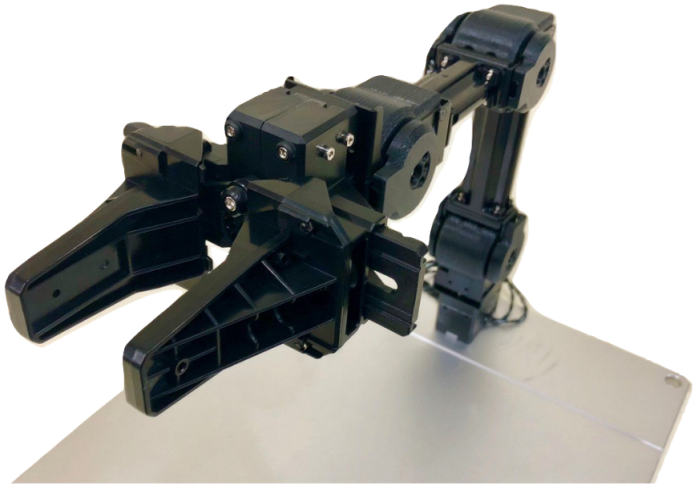
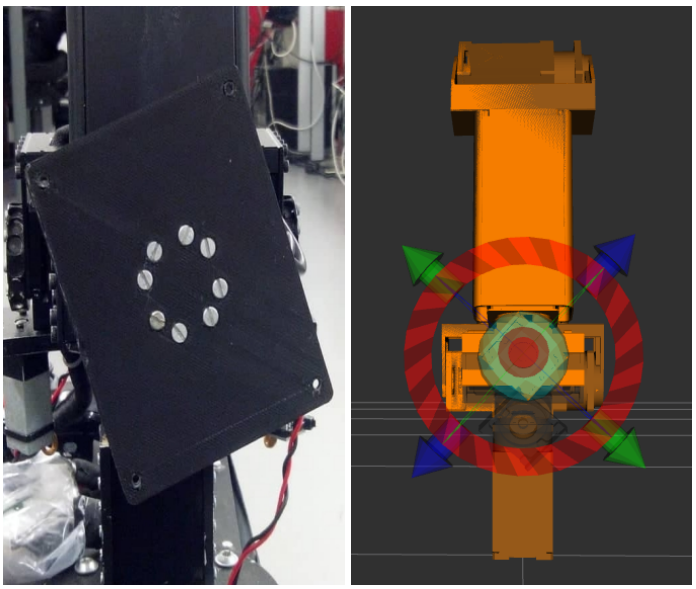


Figura 5.15: *Open Manipulator Chain* da *Robotis*

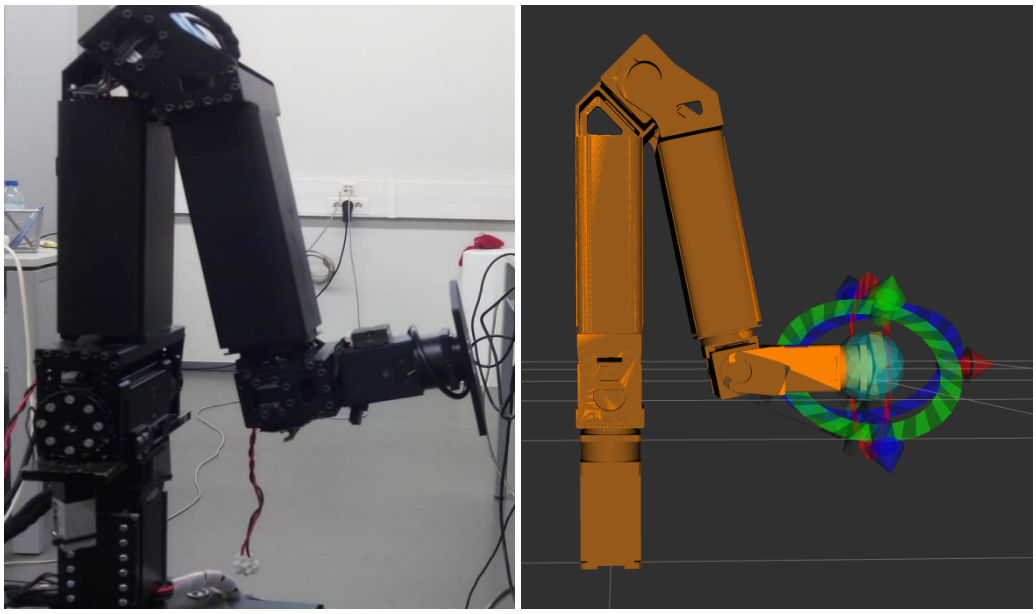
forma a compatibilizá-lo com o *Robotis Manipulator-H*. Desta feita, e após um esforço de cerca de 1 mês na adaptação do código, foi de facto possível conectar o *MoveIt!*.

Foi apenas detetado um problema na implementação após a adaptação do código. Quando foram efetuados os primeiros testes no manipulador, usou-se o *GUI* do *MoveIt!* no *RViz* de forma a mover o manipulador para posições passíveis de serem configuradas a olho nu. Ao fim de alguns testes, foi percetível que as posições da 5^a e 6^a juntas do manipulador não estavam concordantes com as imagens apresentadas na janela *RViz*. Um exemplo deste caso é mostrado nas figuras 5.16 e 5.17, onde é possível distinguir claramente que estas duas juntas não apresentam na realidade as posições indicadas pelo *MoveIt!*.



(a) (b)

Figura 5.16: Vista frontal da posição a) real do manipulador b) devolvida pelo *MoveIt!*



(a) (b)

Figura 5.17: Vista lateral da posição a) real do manipulador b) devolvida pelo *MoveIt!*

Não foi possível averiguar a causa certa para este fenómeno, ainda que uma possível justificação pudesse ser o facto do *package open_manipulator_dynamixel_ctrl*, responsável por efetuar a *interface* entre o *MoveIt!* e os motores, ter sido originalmente desenvolvida para controlar as 4 juntas do *Open Manipulator Chain*, pelo que não estaria preparado para ler os valores das duas restantes juntas do *Manipulator-H*. Ainda assim, as leituras estavam a ser efetuadas, pelo que esta opção foi descartada.

Através dos *packages* do *Manipulator-H* foi possível verificar que os valores devolvidos pelo *MoveIt!* em relação à posição das juntas do manipulador de facto correspondiam aos devolvidos para a posição do manipulador no *GUI* do *MoveIt!* no *RViz*. Assim, foi descartada a possibilidade do erro provir dos motores. Este *package* não podia no entanto ser utilizado para interagir com o *MoveIt!*, uma vez que o modo como eram enviados os comandos de posição não era compatível com a implementação dos *packages* do *Open Manipulator Chain*.

A solução para o problema surgiu de um projeto anterior a esta dissertação e que envolveu este manipulador. Ainda que o *MoveIt!* não tivesse sido usado nesse projeto, foram nele desenvolvidas um conjunto de funções que permitiam mover o manipulador para um dado ponto no espaço. A função que movia o manipulador não foi útil para esta dissertação pois apenas implementava trajetórias baseadas em curvas de *Bézier*[7] de primeira ordem, isto é, linhas retas, além de que não consideravam obstáculos intransponíveis no planeamento. No entanto, foi desenvolvida uma função que permitia devolver as posições das juntas do manipulador. Como esta função devolvia os mesmos valores das juntas que devolviam os *packages* originais do *Manipulator-H* e como era compatível com as funções dos *packages* do *Open Manipulator Chain*, foi usada na solução

[7] https://en.wikipedia.org/wiki/B%C3%A9zier_curve

final implementada. A figura capítulo 5.18 apresenta a posição *home* do manipulador, possível de atingir com esta implementação.

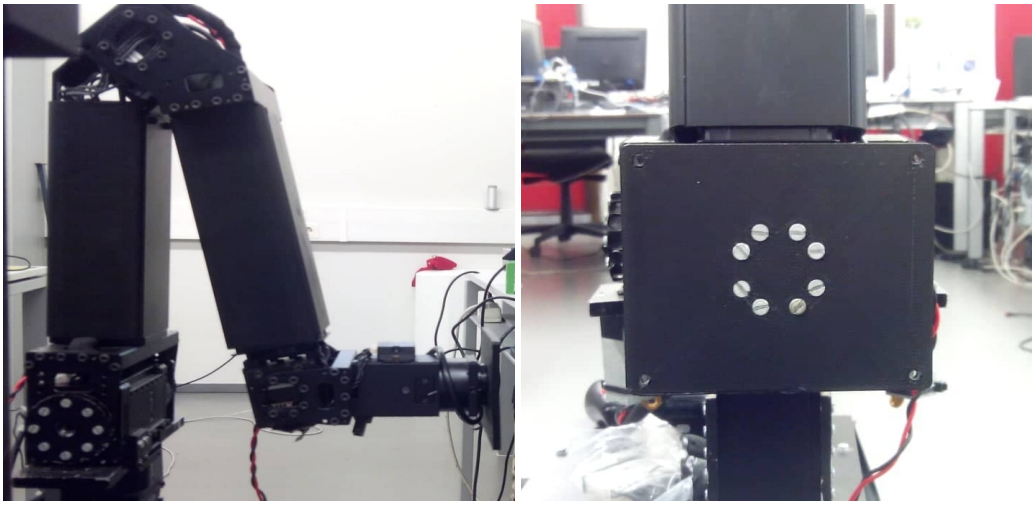


Figura 5.18: Vistas lateral e frontal do manipulador posicionado na sua posição *home* através do *MoveIt!*

Capítulo 6

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas as conclusões finais sobre os resultados da dissertação. É também efetuada uma reflexão sobre o cumprimento das tarefas delineadas e do objetivo principal definido. Por último, é também indicado o trabalho futuro que se sugere.

6.1 Conclusões

O tema desta dissertação visa encontrar uma solução para o planeamento de trajetórias de um braço robótico que permita automatizar a poda das vinhas da encosta do Douro. Como modo de contextualização foi inicialmente efetuada uma pesquisa acerca da poda em Portugal e das épocas em que se realiza. Uma vez que é uma atividade essencial em culturas vinícolas e que cada vez mais carece de mão de obra especializada, justifica-se automatização deste processo. No sua essência, um sistema de poda automatizada pode dividir-se em duas grandes vertentes: a deteção e seleção das vides a podar e o método como é efetuada a sua poda. Esta dissertação foca-se apenas no método de poda.

Foram pesquisadas soluções de poda automática já existentes tendo sido encontradas quatro, onde três delas utilizam manipuladores com 6 graus de liberdade. Relativamente às ferramentas de corte, a mais frequente consiste na adaptação elétrica da tesoura de poda típica. A solução apresentada para a ferramenta de poda consiste precisamente numa tesoura deste tipo, enquanto que o robô de poda disponibilizado foi o manipulador *Robotis Manipulator-H*, também com 6 graus de liberdade, que foi montado na plataforma robótica AGROB V16.

Uma vez selecionado o *hardware*, passou-se à pesquisa de soluções de planeamento de trajetórias, a serem executadas pelo braço robótico no processo de poda. Todas os projetos analisados usam algoritmos disponibilizados na biblioteca *OMPL*, que apresenta 23 planeadores implementados no *software MoveIt!*. Este programa é compatível com *ROS* e direcionado para o planeamento de trajetórias de manipuladores tendo em conta restrições espaciais como objetos intransponíveis, como é o caso das vides das vinhas que apresentam disposições aleatórias. Este programa foi

selecionado como a solução para o planeamento de trajetórias, a fim de avaliar a melhor combinação de algoritmos e respetivas configurações que permitam levar a tesoura de poda montada no manipulador até à posição de corte e de volta à sua posição original.

Para a avaliação dos algoritmos foi desenvolvido em *Gazebo* um cenário de simulação constituído por um modelo da plataforma AGROB V16 e por um modelo de vinha. Foram utilizados 2 modelos de vinha, um simples e um complexo, sendo que o segundo se distingue do primeiro por conter troncos dispostos em direções aleatórios. O *MoveIt!* foi integrado na plataforma de modo a mapear a vinha, utilizando o *laser* na sua torre que lhe permite identificá-la como sendo um objeto de colisão a evitar aquando do planeamento de trajetórias.

A implementação dos planeadores no *MoveIt!* dispõe de 36 parâmetros configuráveis, sendo apenas o parâmetro *longest_valid_segment_fraction*, comum entre todos. O seu valor dita a distância entre *waypoints* de uma trajetória e é fulcral neste tipo de aplicações, onde é necessária uma precisão da trajetória na ordem de valores dos diâmetros das vides, ou seja, centimétrica ou até milimétrica. Por esta razão, este foi o único parâmetro alterado.

Foram efetuados 5 testes com o modelo de vinha simples. Os 4 primeiros consistiam em alterar o valor deste parâmetro de forma a estudar a sua influência nos resultados obtidos para os movimentos que consistiam em levar a tesoura de poda da sua posição original para um ponto de corte e deste ponto para a sua posição original. Os melhores resultados foram posteriormente avaliados no 5º teste de modo a combinar os algoritmos, e respetivos valores do parâmetro *longest_valid_segment_fraction*, que melhores valores temporais e de assertividade tiveram em ambos os movimentos.

Os melhores resultados foram obtidos com o valor de $1cm$ neste parâmetro. Os algoritmos *BiEST*, *LBKPIECE* e *LazyPRMstar* no primeiro movimento, e os algoritmos *BiEST* e *KPIECE* no segundo, foram selecionado para ao 5º teste, tendo obtido os melhores valores médios de tempo de planeamento, além de que não chocaram com a vinha em 17 de 20 vezes. Concluiu-se ainda que o valor predefinido do parâmetro, $5cm$, é excessivamente elevado para a aplicação em causa, onde os diâmetros das vides não tendem a ultrapassar os $2cm$. Devido a este facto, as trajetórias calculadas não apresentam o número de *waypoints* necessário que permita atingir o detalhe exigido na zona de aproximação da sua posição de corte final. Assim, valores altos deste parâmetro geraram mais colisões com a vinha. Por outro lado, quanto mais baixos forem os seus valores (foram testados valores até $1mm$), menos colisões existem, aumentando também os casos de trajetórias bem planeadas e daquelas que não foram computadas em tempo útil, uma vez que requerem tempos de planeamento superiores aos limites definidos.

O 5º teste consistia em movimentar a plataforma ao longo da vinha de modo a que fosse possível executar os mesmo dois movimentos em 5 pontos espalhados ao longo dela. Neste teste foram combinados os algoritmos acima referidos, sempre com o valor de $1cm$ no parâmetro. As configurações que usaram o algoritmo *LazyPRMstar* no primeiro movimento obtiveram os melhores resultados, nunca tendo colidido com a vinha quando o algoritmo *BiEST* foi usado no segundo movimento, e tendo colidido uma vez com o algoritmo *KPIECE*, num total de 10 testes realizados. Estas combinações também foram as que melhores valores temporais obtiveram, pelo que foram

as únicas selecionadas para o último teste.

Este último teste foi realizado no modelo de vinha complexa, onde foram testadas as duas configurações acima referidas num conjunto de 3 pontos com posições e orientações típicas dos pontos de corte numa vinha. Nos 10 testes realizados, a combinação que usou o algoritmo *BiEST* foi no segundo movimento conseguiu computar corretamente 9 planos, ao passo que a que usou o algoritmo *KPIECE* apenas foi capaz de o fazer 3 vezes. A combinação com o os algoritmos *LazyPRMstar* e *BiEST* no primeiro e segundo movimentos, respetivamente, revela-se como sendo a que melhor se ajusta para o planeamento de trajetórias para pontos de corte em vinhas. O valor temporal médio esperado para calcular os dois movimentos exigidos para o corte de uma vide (e não na execução dos movimentos), com esta combinação, é de 6,456s.

Nos dois últimos testes foi ainda possível avaliar em que posições é que a plataforma se encontrava sempre que os algoritmos conseguiam computar planos para o primeiro movimento. Verificou-se que, independentemente do algoritmo, alguns pontos exigiam que o AGROB V16 se encontrasse numa posição específica para permitir o planeamento de movimentos, não bastando apenas que este estivesse ao alcance do manipulador. Desta forma, destaca-se a posição da plataforma como determinante no planeamento de movimentos, não bastando assim que esta se encontre em frente ao ponto. Ainda assim, para cada ponto há uma região onde se pode encontrar a plataforma, que garanta que o planeamento dos movimentos é possível. Ainda que não tenha sido testado, é possível que pequenas inclinações, caraterísticas dos ambientes pedregosos das encostas do Douro, não apresentem qualquer entrave ao planeamento, desde que a plataforma se encontre nesta região e que os pontos estejam ao alcance do manipulador.

Por último, há ainda outro detalhe a destacar nos 2 últimos testes, relacionado com a heurística apresentada para a movimentação da plataforma entre pontos de corte. Esta supõe que o AGROB V16 se movimenta sempre que não é encontrada solução para o planeamento do primeiro movimento, pelo que tempos limite excessivamente altos diminuem a performance da plataforma. Assim, a escolha do algoritmo *LazyPRMstar*, com um tempo constante de 2s para o planeamento do primeiro movimento, revela-se novamente a melhor escolha. Os restantes algoritmos, que não são limitados temporalmente, além de serem mais lentos a computar soluções, fariam com que a plataforma se movesse apenas de 15 em 15s, uma vez que este é o tempo limite que têm para o cálculo de trajetórias.

6.2 Satisfação dos Objetivos

No início desta dissertação foram definidas 4 principais tarefas para o cumprimento do principal objetivo: (1) Escolha de uma ferramenta de planeamento de trajetórias para a integração com o braço robótico, (2) Desenvolvimento do ambiente simulado onde foram efetuados os testes, (3) Avaliação, em ambiente simulado, de algoritmos para o planeamento de trajetórias em aplicações de vinha e (4) Avaliação, em ambiente real, de algoritmos para o planeamento de trajetórias em

aplicações de vinha. As primeiras três tarefas foram cumpridas, sendo que a última não foi concretizada pelo facto da tesoura de poda não ter sido disponibilizada para a realização dos testes. Assim, e tendo em conta os resultados obtidos, considera-se cumprido o objetivo desta dissertação.

Reconhece-se, no entanto, que a aplicação do *MoveIt!* em atividades de poda de vinhas carece de um estudo mais aprofundado. Futuras melhorias são de seguidas enunciadas.

6.3 Trabalho Futuro

Como melhorias futuras, sugerem-se as seguintes:

- A inclusão das lâminas da tesoura de poda no planeamento de trajetórias, de modo a que se consigam trajetórias em que as lâminas se encontrem fechadas até regiões próximas dos pontos de corte, e só aí sejam abertas;
- Testar a possibilidade da inclusão de um ponto intermédio no planeamento que se encontre próximo do ponto final de corte mas cujo trajeto, desde a posição original do planeador até esse ponto, se encontre desimpedido, de modo a que algoritmos que computem soluções mais rapidamente possam ser usados para o cálculo desta trajetória livre de obstáculos;
- Avaliar com maior profundidade os parâmetros individuais de cada algoritmo;
- Criar um mapa de pontos de uma das vinhas de encosta, de modo a que os testes sejam realizados num modelo equivalente aos que serão encontrados num ambiente real;
- Criar na simulação um ambiente pedregoso que mais se assemelhe ao ambiente real das vinhas;
- Testar o planeamento de trajetórias com o AGROB V16 rodado ligeiramente sobre si mesmo, uma vez que no ambiente real ele certamente não se irá encontrar disposto paralelamente à vinha.

Referências

[1] M. Almeida. *O ciclo do vinho*. Setembro de 2015. URL: http://mariajoaodealmeida.com/2015/09/o-ciclo-do-vinho/. (acedido em: 20.1.2018).

[2] B. Aguiar. *Poda de árvores e arbustos*. Fevereiro de 2014. URL: https://lifestyle.sapo.pt/casa-e-lazer/dicas-e-decoracao/artigos/poda-de-arvores-e-arbustos. (acedido em: 20.1.2018).

[3] C. Almeida. *Poda da videira*. URL: http://www.drapc.min-agricultura.pt/base/documentos/poda_da_videira.pdf. (acedido em: 20.1.2018).

[4] F. Champagnol. *Éléments de physiologie de la vigne et de viticulture générale*. F. Champagnol, 1984. ISBN: 9782950061409. URL: https://books.google.pt/books?id=jTUyAAAACAAJ.

[5] L. Mendes. *Póvoa de Cervâes - A Terra e as Gentes*. 2001. (acedido em: 20.1.2018).

[6] Q. Zhang e F.J. Pierce. *Agricultural Automation: Fundamentals and Practices*. Taylor & Francis, 2013. ISBN: 9781439880579. URL: https://books.google.pt/books?id=qYvThP8PoqkC.

[7] C.W. Bac et al. "Harvesting Robots for High-value Crops: State-of-the-art Review and Challenges Ahead". Em: *Journal of Field Robotics* 31.6 (2014). cited By 71, pp. 888–911. DOI: 10.1002/rob.21525. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84939470657&doi=10.1002%5C%2frob.21525%5C&partnerID=40%5C&md5=ed24b5c3a9cfefef950b44af14df5f26.

[8] Vision Robotics Corporation. *Intelligent Autonomous Grapevine Pruner*. 2015. URL: https://www.visionrobotics.com/vr-grapevine-pruner. (acedido em: 25.1.2018).

[9] T. Botterill et al. "A Robot System for Pruning Grape Vines". Em: *Journal of Field Robotics* 34.6 (2017). cited By 0, pp. 1100–1122. DOI: 10.1002/rob.21680. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84990207231&doi=10.1002%5C%2frob.21680%5C&partnerID=40%5C&md5=ff82e340fe030cfdc65e80a5bfdf94b8

[10] Sam Corbett-Davies et al. "An expert system for automatically pruning vines". Em: *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*. ACM. 2012, pp. 55–60.

[11] Scott Paulin et al. “A specialised collision detector for grape vines”. Em: *Proceedings of the Australasian Conference on Robotics and Automation*. 2015, pp. 1–5.

[12] A Bechar et al. “Visual Servoing Methodology for Selective Tree Pruning by Human-Robot Collaborative System”. Em: (2014).

[13] Wall-Ye. 2018. URL: http://wall-ye.com/index-2.html. (acedido em: 25.1.2018).

[14] “Robot Manipulators and Control Systems”. Em: *Industrial Robots Programming: Building Applications for the Factories of the Future*. Boston, MA: Springer US, 2007, pp. 35–107. ISBN: 978-0-387-23326-0. DOI: 10.1007/978-0-387-23326-0_2. URL: https://doi.org/10.1007/978-0-387-23326-0_2.

[15] M. Bélanger-Barrette. *How to Choose the Right Industrial Robot?* Abril de 2014. URL: https://blog.robotiq.com/bid/70408/How-to-Choose-the-Right-Industrial-Robot.

[16] *IEC 60529:1989+AMD1:1999+AMD2:2013 CSV Consolidated version*. Agosto de 2013. URL: https://webstore.iec.ch/publication/2452.

[17] Scott Paulin et al. “Finding shorter paths for robot arms using their redundancy”. Em: *arXiv preprint arXiv:1708.06067* (2017).

[18] Dezembro de 2017. URL: http://cobotsguide.com/2016/06/universal-robots/.

[19] Robotis. *Robotis Manipulator-H*. URL: http://www.robotis.us/robotis-manipulator-h/.

[20] M. Rose. *End effector*. Fevereiro de 2009. URL: http://whatis.techtarget.com/definition/end-effector.

[21] Skil. *Escolher as ferramentas de poda*. URL: https://www.skil.pt/instrucoes-passo-a-passo/escolher-as-ferramentas-de-poda.html.

[22] Skil. *Podar videiras*. URL: https://www.skil.pt/instrucoes-passo-a-passo/podar-videiras.html.

[23] The Home Depot. *Pruning Tools*. URL: https://www.homedepot.com/c/pruning_tools_HT_BG_OD.

[24] Infaco. *Electrocoup F3015*. URL: https://www.infaco.com/pt/produtos/f3015/ficha-de-produto-f3015.

[25] J.C. Latombe. *Robot Motion Planning*. The Springer International Series in Engineering and Computer Science. Springer US, 2012. ISBN: 9781461540229. URL: https://books.google.pt/books?id=nQ7aBwAAQBAJ.

[26] H.M. Choset. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. A Bradford book. Prentice Hall of India, 2005. ISBN: 9780262033275. URL: https://books.google.pt/books?id=S3biKR21i-QC.

[27] C. Martinez. “A comparison among different sampling-based planning techniques”. Em: (2015).

[28] Peng Cheng. *Sampling-based motion planning with differential constraints*. Rel. téc. 2005.

[29] V.J. Lumelsky e A.A. Stepanov. “Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape”. Em: *Algorithmica* 2.1-4 (1987). cited By 178, pp. 403–430. DOI: 10.1007/BF01840369. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-2842550433&doi=10.1007%5C%2fBF01840369&partnerID=40&md5=6557b173164fa3eba5dc9032091b677f.

[30] Y. K. Hwang e N. Ahuja. “A potential field approach to path planning”. Em: *IEEE Transactions on Robotics and Automation* 8.1 (Fevereiro de 1992), pp. 23–32. ISSN: 1042-296X. DOI: 10.1109/70.127236.

[31] B.H. Krogh e Robotics International of SME. *A Generalized Potential Field Approach to Obstacle Avoidance Control*. Creative manufacturing engineering program. RI/SME, 1984. URL: https://books.google.pt/books?id=SFibtgAACAAJ.

[32] W. Burgard et al. *Robot Motion Planning*. 2011. URL: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf.

[33] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN: 9781139455176. URL: https://books.google.pt/books?id=-PwLBAAAQBAJ.

[34] A. Sucan. “Task and motion planning for mobile manipulators”. Tese de doutoramento. Rice University, 2011.

[35] L. E. Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. Em: *IEEE Transactions on Robotics and Automation* 12.4 (Agosto de 1996), pp. 566–580. ISSN: 1042-296X. DOI: 10.1109/70.508439.

[36] Franz Aurenhammer. “Voronoi diagrams—a survey of a fundamental geometric data structure”. Em: *ACM Computing Surveys (CSUR)* 23.3 (1991), pp. 345–405.

[37] Howie Choset e Philippe Pignon. “Coverage Path Planning: The Boustrophedon Cellular Decomposition”. Em: *Field and Service Robotics*. Ed. por Alexander Zelinsky. London: Springer London, 1998, pp. 203–209. ISBN: 978-1-4471-1273-0.

[38] H. Choset. *Robotic motion planning: Cell decompositions*. URL: https://www.cs.cmu.edu/~motionplanning/lecture/Chap6-CellDecomp_howie.pdf.

[39] M. Elbanhawi e M. Simic. “Sampling-Based Robot Motion Planning: A Review”. Em: *IEEE Access* 2 (2014), pp. 56–77. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2014.2302442.

[40] Martin Barland. “Motion Planning Framework for Industrial Manipulators using the Open Motion Planning Library (OMPL)”. Tese de mestrado. Institutt for teknisk kybernetikk, 2012.

[41] Scott Paulin et al. “Integrating asymptotically-optimal path planning with local optimization”. Em: *CoRR* abs/1708.06056 (2017). arXiv: 1708.06056. URL: http://arxiv.org/abs/1708.06056.

[42] E. Theodorou, J. Buchli e S. Schaal. “Reinforcement learning of motor skills in high dimensions: A path integral approach”. Em: *2010 IEEE International Conference on Robotics and Automation*. Maio de 2010, pp. 2397–2403. DOI: 10.1109/ROBOT.2010.5509336.

[43] N. Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. Em: *2009 IEEE International Conference on Robotics and Automation*. Maio de 2009, pp. 489–494. DOI: 10.1109/ROBOT.2009.5152817.

[44] Ioan A. Şucan, Mark Moll e Lydia E. Kavraki. “The Open Motion Planning Library”. Em: *IEEE Robotics & Automation Magazine* 19.4 (Dezembro de 2012). http://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[45] R. Bohlin e L. E. Kavraki. “Path planning using lazy PRM”. Em: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. Abril de 2000, 521–528 vol.1. DOI: 10.1109/ROBOT.2000.844107.

[46] Sertac Karaman e Emilio Frazzoli. “Sampling-based Algorithms for Optimal Motion Planning”. Em: *CoRR* abs/1105.1186 (2011). arXiv: 1105.1186. URL: http://arxiv.org/abs/1105.1186.

[47] K. Hauser. “Lazy collision checking in asymptotically-optimal motion planning”. Em: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Maio de 2015, pp. 2951–2957. DOI: 10.1109/ICRA.2015.7139603.

[48] C. Nissoux, T. Simeon e J. P. Laumond. “Visibility based probabilistic roadmaps”. Em: *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*. Vol. 3. 1999, 1316–1321 vol.3. DOI: 10.1109/IROS.1999.811662.

[49] Andrew Dobson, Athanasios Krontiris e Kostas E Bekris. “Sparse roadmap spanners”. Em: *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 279–296.

[50] L. Jaillet, J. Cortés e T. Siméon. “Sampling-Based Path Planning on Configuration-Space Costmaps”. Em: *IEEE Transactions on Robotics* 26.4 (Agosto de 2010), pp. 635–646. ISSN: 1552-3098. DOI: 10.1109/TRO.2010.2049527.

[51] J.C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2005. ISBN: 9780471441908. URL: https://books.google.pt/books?id=f66OIvvkKnAC.

[52] D. Hsu, J. C. Latombe e R. Motwani. “Path planning in expansive configuration spaces”. Em: *Proceedings of International Conference on Robotics and Automation*. Vol. 3. Abril de 1997, 2719–2726 vol.3. DOI: 10.1109/ROBOT.1997.619371.

[53] I. A. Şucan e L. E. Kavraki. “A Sampling-Based Tree Planner for Systems With Complex Dynamics”. Em: *IEEE Transactions on Robotics* 28.1 (Fevereiro de 2012), pp. 116–131. ISSN: 1552-3098. DOI: 10.1109/TRO.2011.2160466.

[54] Sergey Brin. “Near neighbor search in large metric spaces”. Em: (1995).

[55] B. Gipson, M. Moll e L. E. Kavraki. “Resolution Independent Density Estimation for motion planning in high-dimensional spaces”. Em: *2013 IEEE International Conference on Robotics and Automation*. Maio de 2013, pp. 2437–2443. DOI: 10.1109/ICRA.2013.6630908.

[56] Lucas Janson e Marco Pavone. “Fast Marching Trees: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions - Extended Version”. Em: *CoRR* abs/1306.3532 (2013). arXiv: 1306.3532. URL: http://arxiv.org/abs/1306.3532.

[57] Joseph A. Starek et al. “An Asymptotically-Optimal Sampling-Based Algorithm for Bi-directional Motion Planning”. Em: *CoRR* abs/1507.07602 (2015). arXiv: 1507.07602. URL: http://arxiv.org/abs/1507.07602.